ASP.NET MVC 5 入门指南

wizardforcel

Published with GitBook



目錄

ASP.NET MVC 5 入门指南

原文：Getting Started with ASP.NET MVC 5

来源：ASP.NET MVC 5 入门指南汇总

本教程将使用Visual Studio 2013手把手教你构建一个入门的ASP.NET MVC 5 Web应用程序。本教程配套的C#源码工程可通过如下网址下载：C#版本源码链接。同时，请查阅 Building the Chapter Downloads 来完成编译源码和配置数据库。

在本教程中的源码工程，您可在Visual Studio中可运行MVC 5应用程序。您也可以使Web应用程序部署到一个托管服务提供商上。微软提供免费的网络托管多达10个网站，free Windows Azure trial account。本教程由Scott Guthrie (twitter @scottgu), Scott Hanselman (twitter: @shanselman), and Rick Anderson (@RickAndMSFT)共同写作完成，由葡萄城控件技术团队（新浪微博 @葡萄城控件）翻译编辑发布。

ASP.NET MVC 5 - 开始MVC5之旅

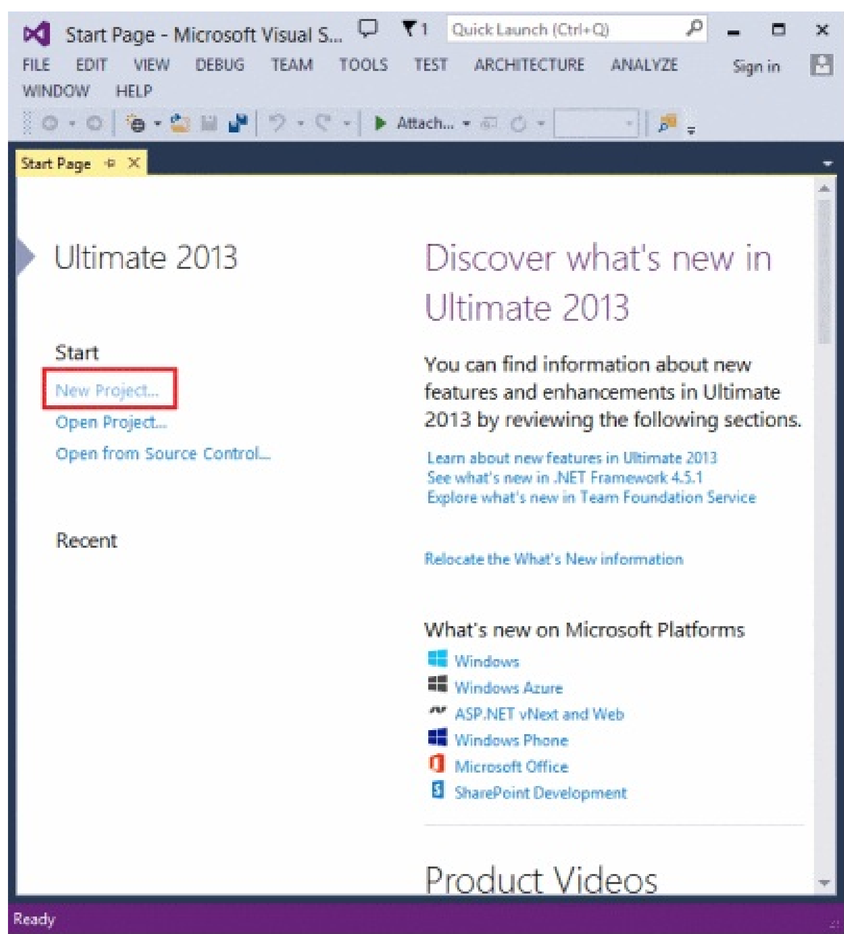
本教程将使用Visual Studio 2013手把手教你构建一个入门的ASP.NET MVC5 Web应用程序。本教程配套的C#源码工程可通过如下网址下载：C#版本源码链接。同时，请查阅 Building the Chapter Downloads 来完成编译源码和配置数据库。

在本教程中的源码工程，您可在Visual Studio中运行MVC 5应用程序。您也可以使Web应用程序部署到一个托管服务提供商上。微软提供免费的网络托管多达10个网站，free Windows Azure trial account。本教程由Scott Guthrie (twitter @scottgu), Scott Hanselman (twitter: @shanselman), and Rick Anderson (@RickAndMSFT)共同写作完成，由葡萄城控件团队翻译（新浪微博 @葡萄城控件）。

入门

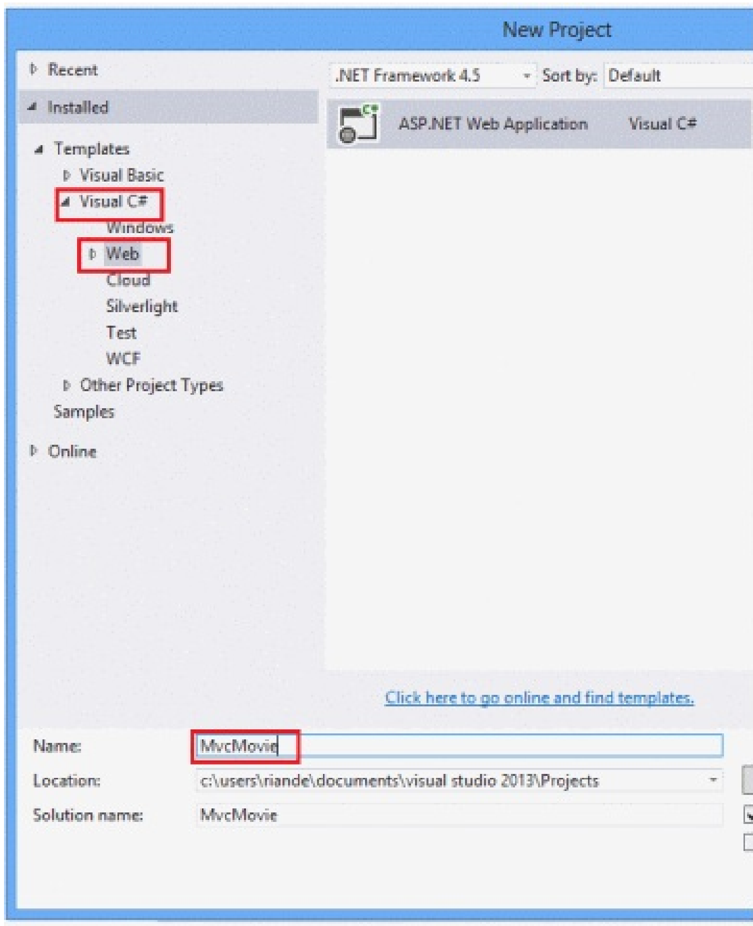
运行 Visual Studio Express 2013 for Web 或 Visual Studio 2013开始这个实例。

Visual Studio是一个IDE集成开发环境。就像您使用Microsoft Word来编写文档，你可以使用集成开发环境（IDE）来创建一个应用程序。在Visual Studio中的一个顶部工具栏中显示了各种不同的选项来供您使用。在IDE中还有一个菜单，提供了另一种方式来执行任务。（例如，您可以不从“**开始**”页面中，选择“**新建项目**”，您可以使用该菜单，然后选择“ 文件**“>“新建项目**“）

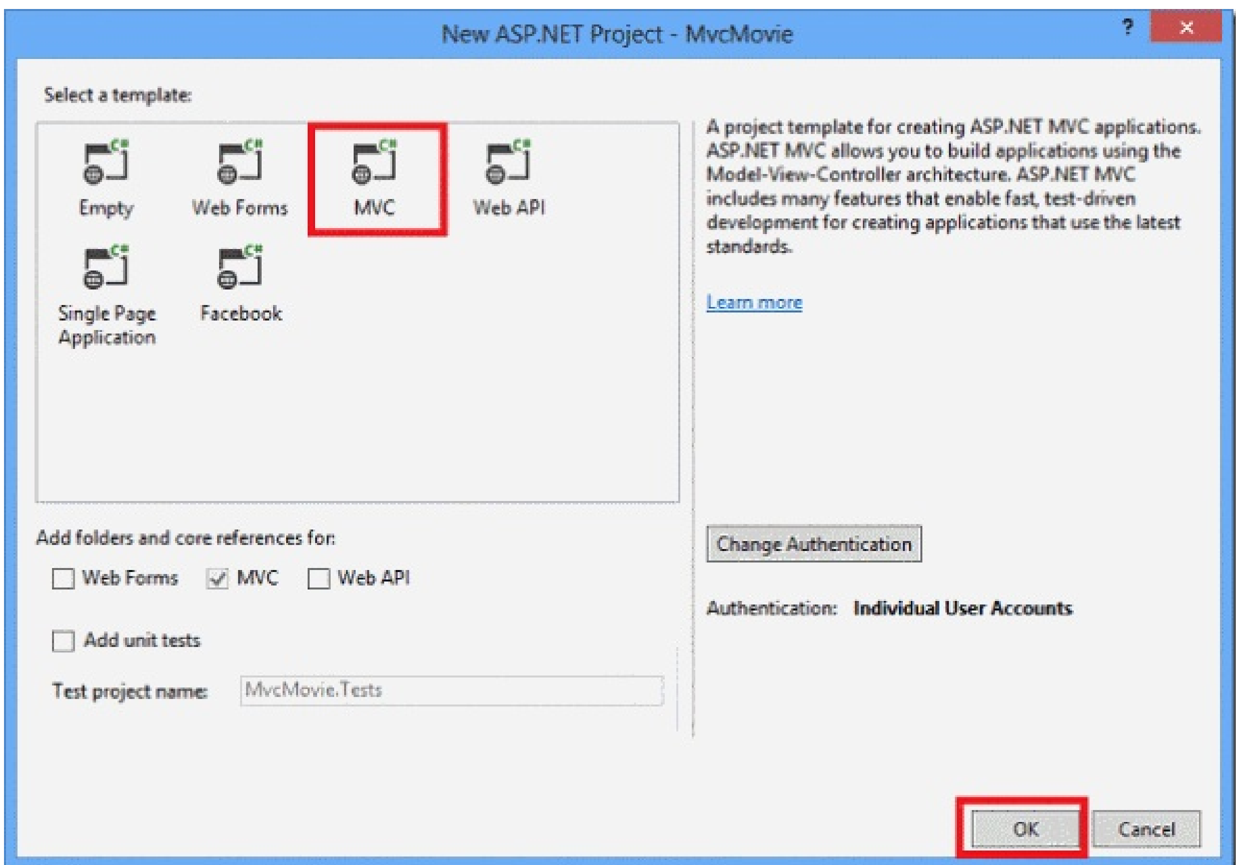


创建您的第一个MVC 5应用程序

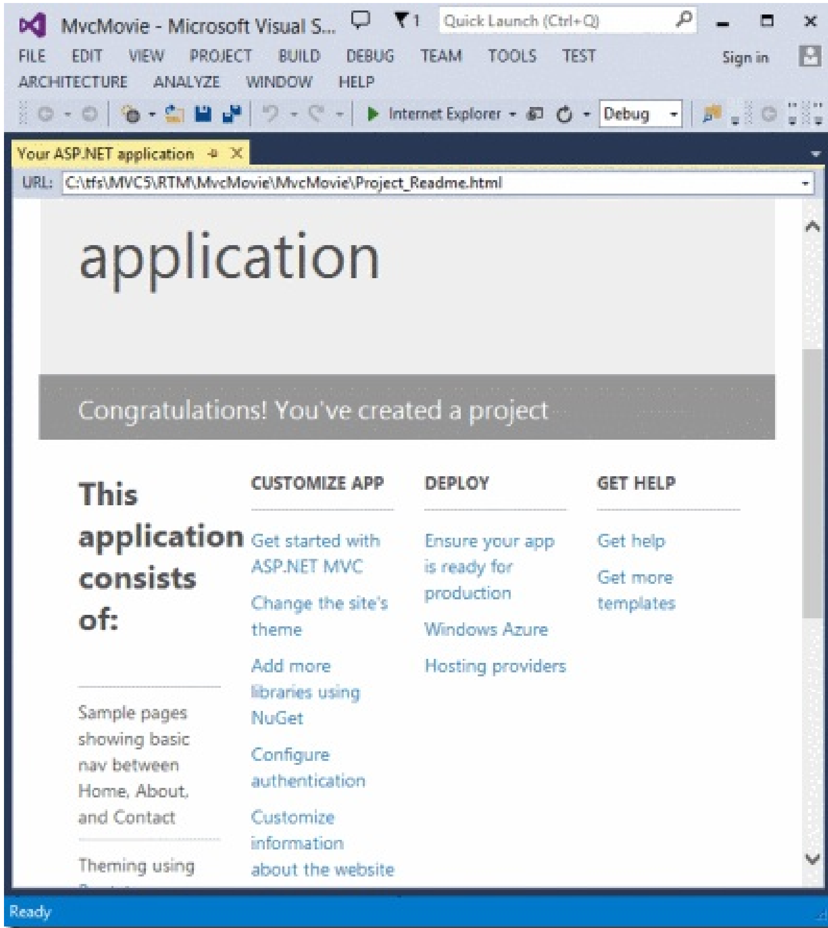
点击新建工程，在左侧选择**Visual C#**， 接着选择**Web**，然后选择**ASP.NET Web Application**. 命名您的工程为"MvcMovie"，然后单击确定.



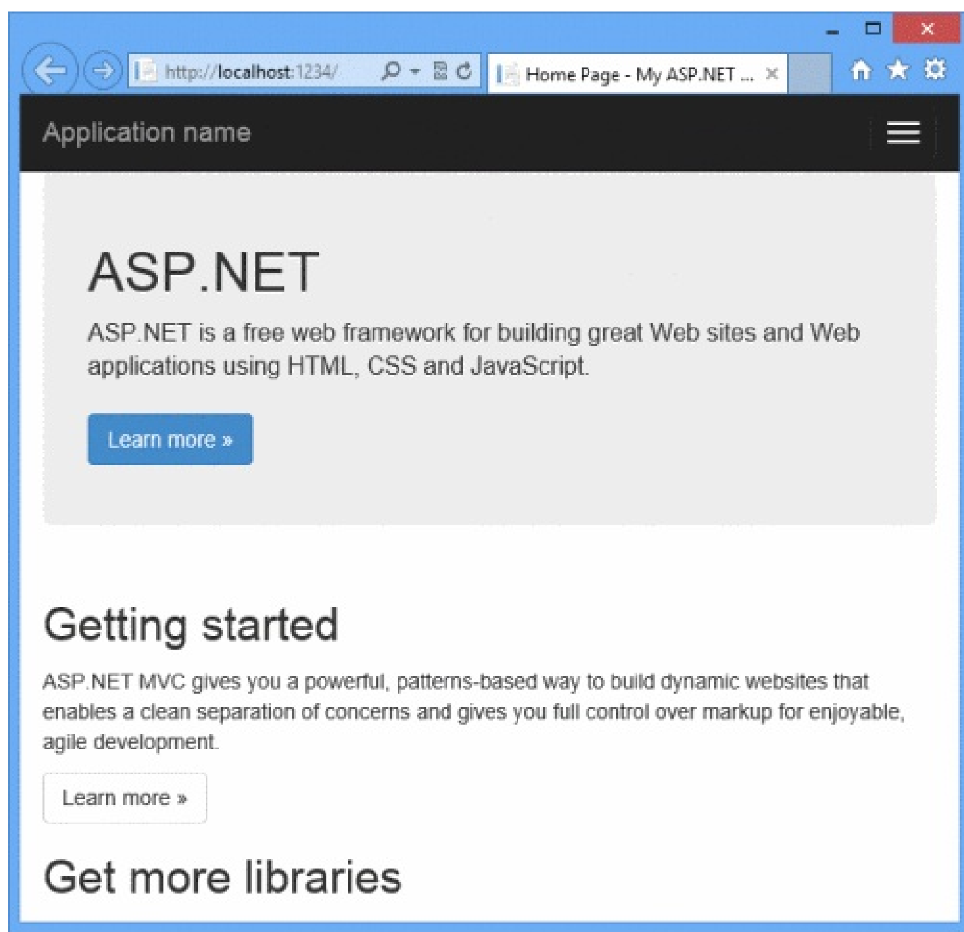
在 **New ASP.NET Project** 对话框，选择 **MVC**模板，然后单击确定。



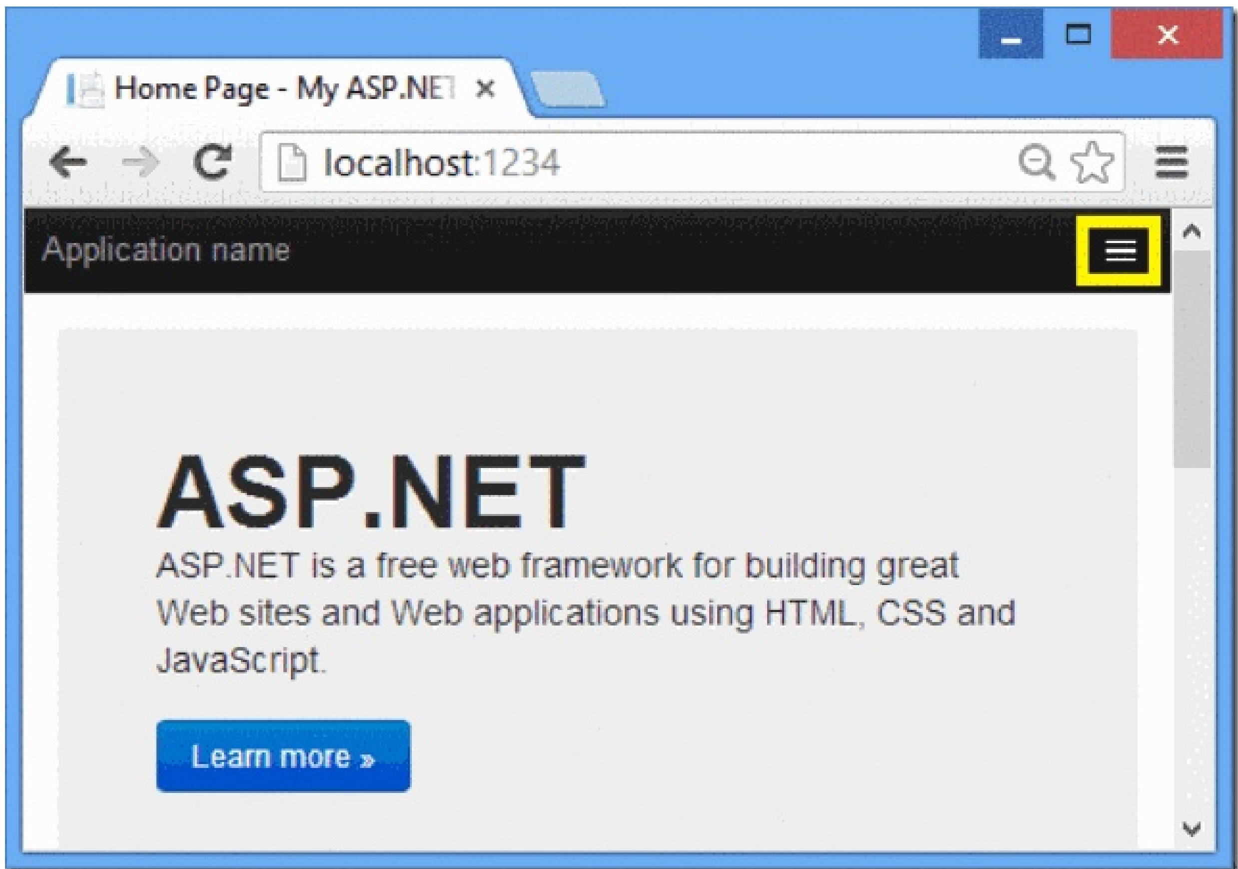
Visual Studio 刚刚创建的 ASP.NET MVC 项目使用了默认的模板，所以在当前的工程中您不需要做任何事情！这是一个简单的"Hello World ！"工程，并且这也是您开始“MvcMovie”工程的好地方。.

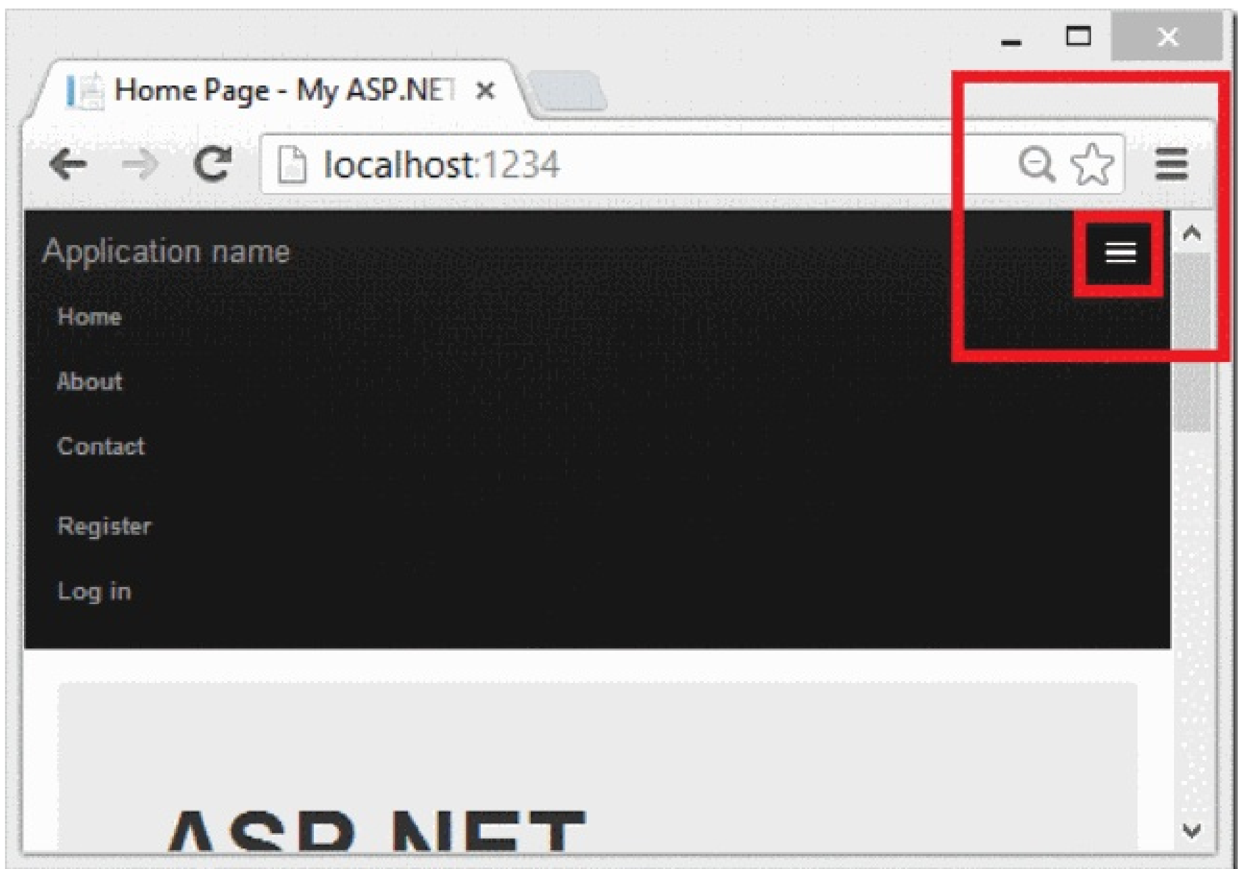


按下键盘快捷键 F5开始启动调试。 F5使得Visual Studio启动IIS Express并运行Web应用程序。然后Visual Studio会启动浏览器并打开应用程序的主页面。请注意，在浏览器的地址栏中会显示localhost:port#而不是像example.com 这样的地址。这是因为 localhost总是会被解析为您自己的本地计算机，在这种情况下，这正是您你刚刚建立的应用程序。当Visual Studio运行一个Web工程时，会使用一个随机端口的Web服务。在下面的图片中，端口号是1234。当您运行该应用程序时，您可能会看到一个不同的端口号。



在默认模板页面的右边，为您提供了“主页(Home)”, “关于(About)”和“联系(Contact)”页面。下面的截图没有看到“主页(Home)”, “关于(About)”和“联系(Contact)”连接。这取决于你浏览器窗体的大小，你可通过点击右上角导航图标看到这些链接。





同时，默认模板创建的ASP.Net MVC应用程序还提供了注册和登录功能。 接下来的一步是修改此默认应用程序，并了解一些关于ASP.NET MVC的知识。关闭浏览器，让我们修改一些源代码吧。

ASP.NET MVC 5 - 控制器

MVC代表: 模型-视图-控制器 。MVC是一个架构良好并且易于测试和易于维护的开发模式。基于MVC模式的应用程序包含：

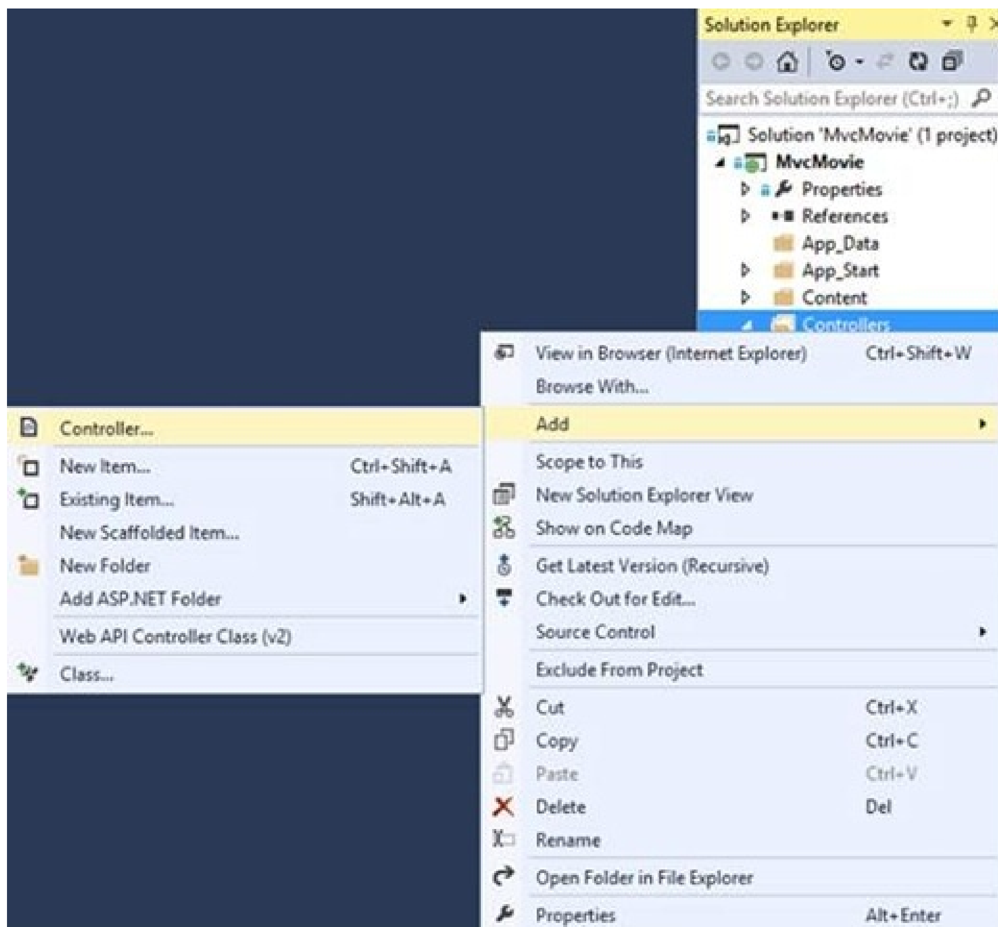
· **M**odels： 表示该应用程序的数据并使用验证逻辑来强制实施业务规则的数据类。

· **V**iews： 应用程序动态生成 HTML所使用的模板文件。

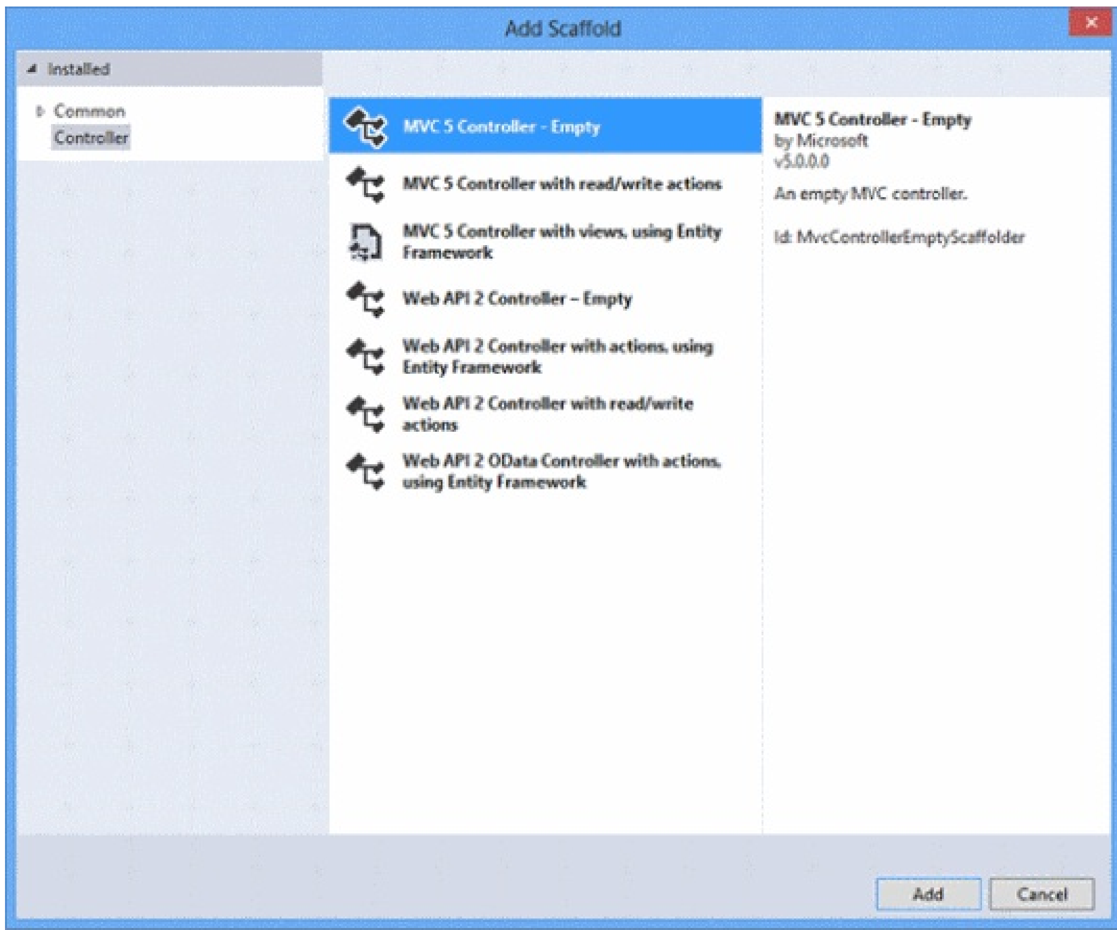
· **C**ontrollers： 处理浏览器的请求，取得数据模型，然后指定要响应浏览器请求的视图模板。

本系列教程，我们将覆盖所有这些概念，并告诉您如何在ASP.NET MVC 5中使用它们来构建应用程序。

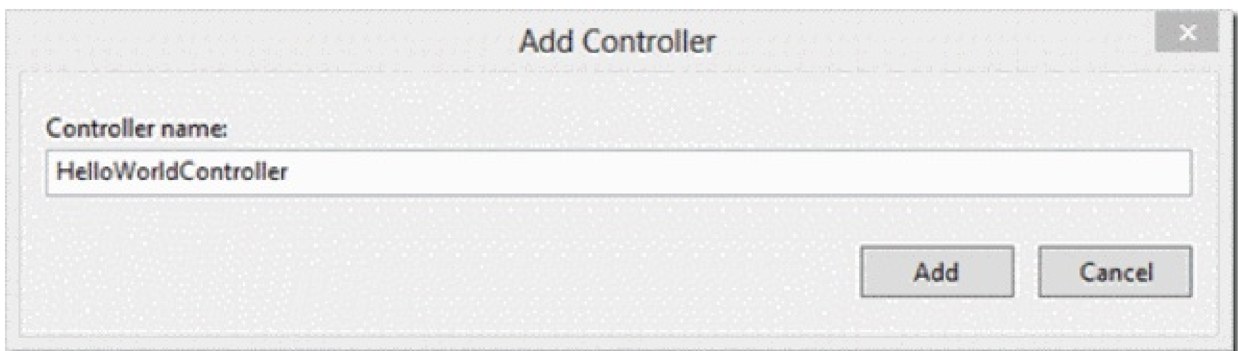
首先，让我们创建一个控制器类。在解决方案资源管理器中，用鼠标右键单击控制器文件夹（*Controllers* ），然后选择“添加控制器“。



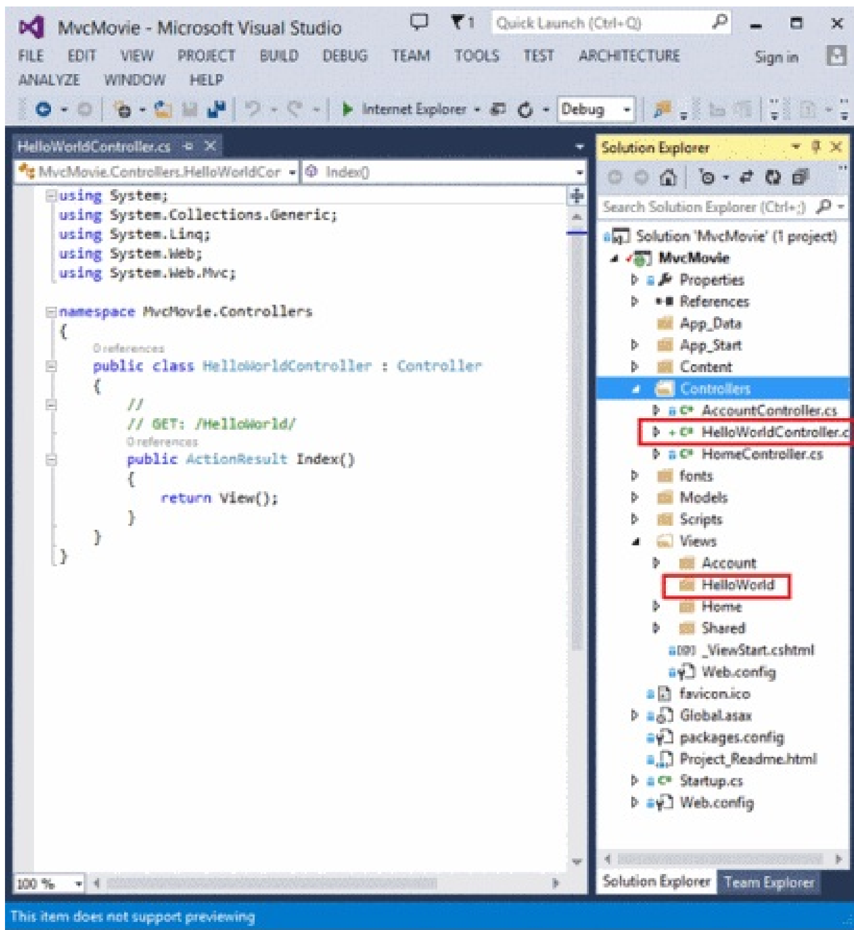
在添加**Scaffold**对话框，单击**MVC5**控制器 - 空，然后单击“添加**”。



命名新的控制器为“HelloWorldController”，并单击“ 添加“。



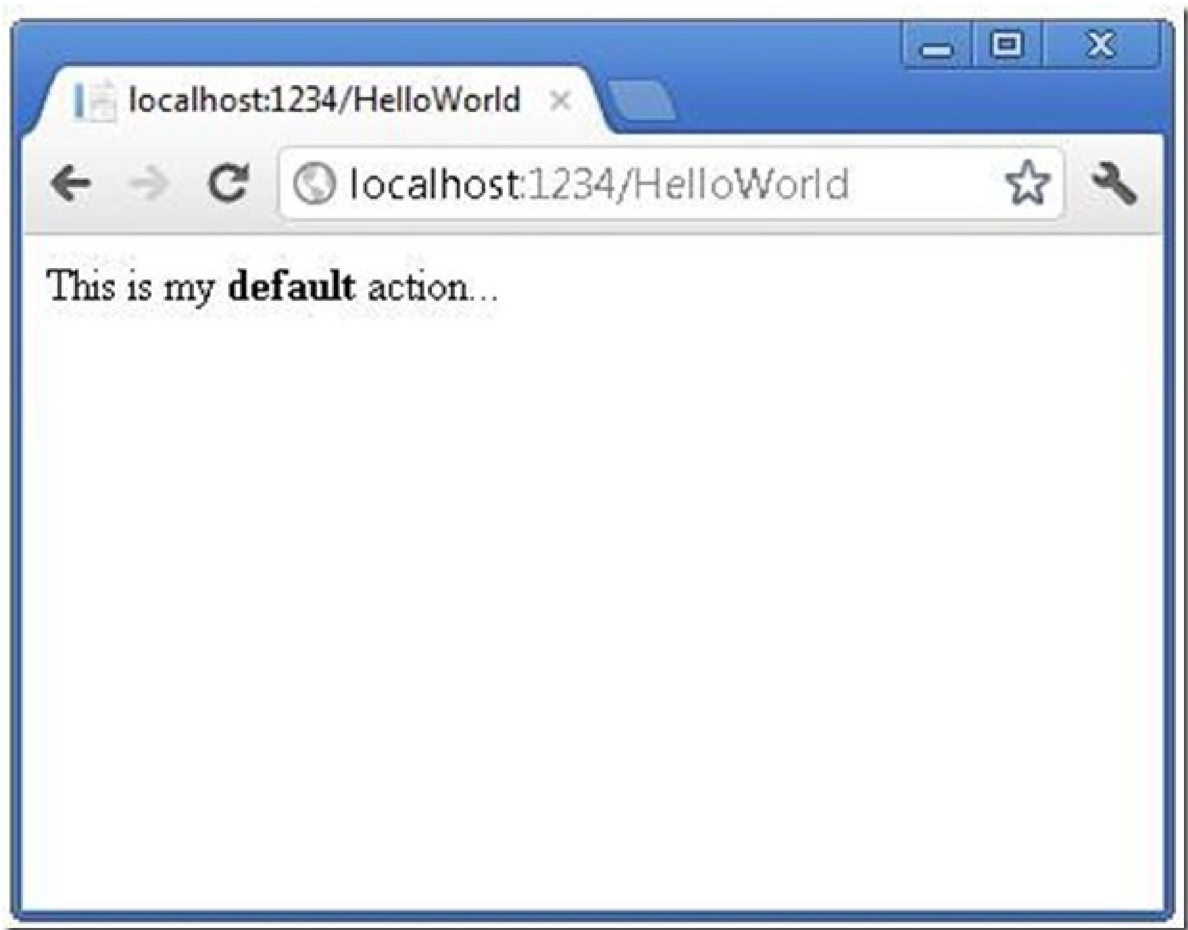
请注意，在解决方案资源管理器中会创建一个名为*HelloWorldController.cs*的新文件和一个新的文件夹*Views\HelloWorld*。该文件会被IDE默认打开。



用下面的代码替换该文件中的内容。

```
using System.Web; using System.Web.Mvc; namespace MvcMovie.Controllers
{ public class HelloWorldController : Controller
    { //
        // GET: /HelloWorld/
        public string Index()
        { return "This is my <b>default</b> action...";
        } //
        // GET: /HelloWorld/Welcome/
        public string Welcome()
        { return "This is the Welcome action method...";
        }
    }
}
```

在这个例子中控制器方法将返回一个字符串的HTML。本控制器被命名HelloWorldController代码中的第一种方法被命名为Index。让我们从浏览器中调用它。运行应用程序（按F5或CTRL + F5）。在浏览器的地址栏中输入路径"HelloWorld"。（例如，在下面的示例中: http://localhost:1234/HelloWorld）页面在浏览器中的表现如下面的截图。在上面的方法中，代码直接返回了一个字符串。你告诉系统只返回一些HTML，系统确实这样做了！



ASP.NET MVC会调用不同的控制器类（和其内部不同的操作方法）这取决于传入URL。所使用的ASP.NET MVC的默认URL路由逻辑使用这样的格式来判定哪些代码以便调用：

/[Controller]/[ActionName]/[Parameters]

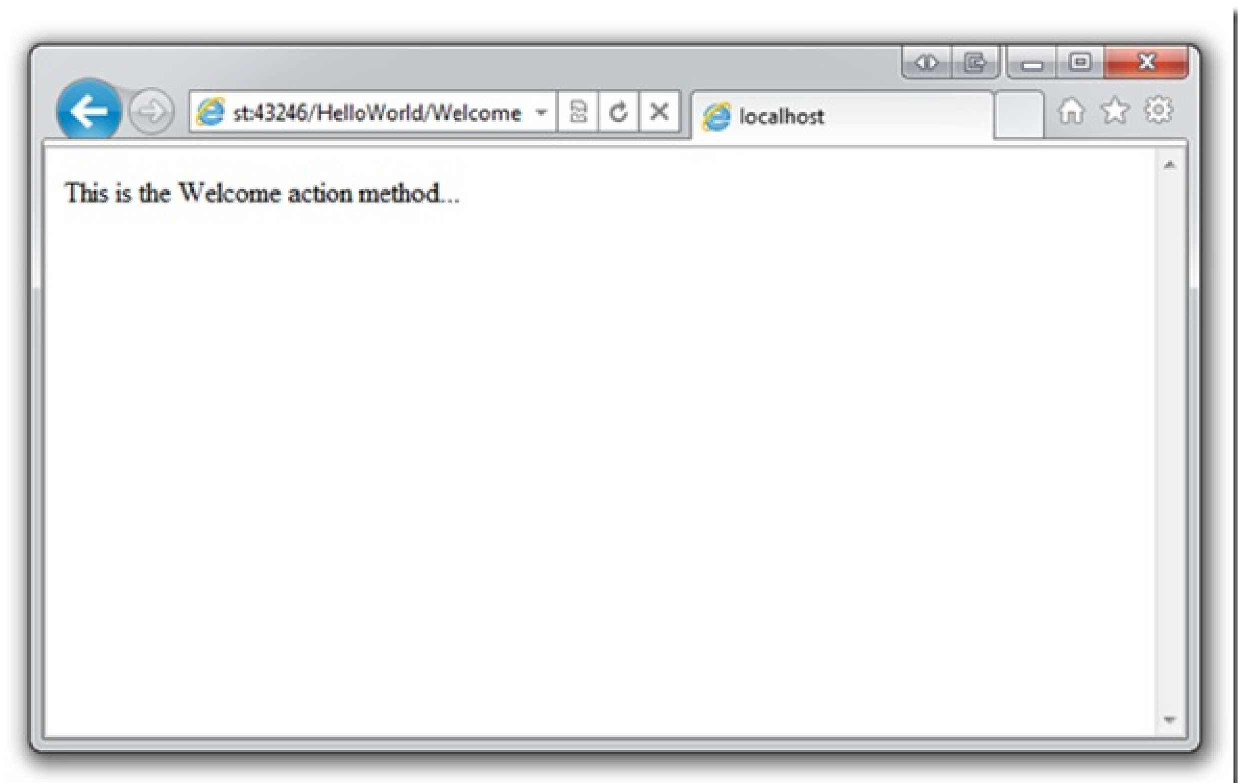
你也可在*App_Start/RouteConfig.cs* 文件内通过配置URL路由解析规则：

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional
);
}
```

如果您运行应用程序并没有提供任何URL段的，默认为“Home”的控制器和“Index”的操作方法，在上面的代码中的defaults部分指定的：

- 第一部分的URL确定哪个控制器类会被执行。因此 */HelloWorld*映射到HelloWorldController控制器类。
- 第二部分的URL确定要执行控制器类中的那个操作方法。因此 */HelloWorld/**Index***，会使得 `HelloWorldController` 控制器类的Index 方法被执行。请注意，我们只需要浏览*/HelloWorld*路径，默认情况下会调用Index方法。如果没有明确的指定操作方法，Index方法会默认的被控制器类调用。
- 第三部分的URL段（Parameters参数）是路由数据。在本教程中，稍后我们将看到路由数据。

浏览_http://localhost:xxxx/HelloWorld/Welcome。_Welcome方法会被运行并返回字符串:"This is the Welcome action method...”。 默认的MVC映射为/[Controller]/[ActionName]/[Parameters] 对于这个URL，控制器类是HelloWorld，操作方法是Welcome，您还没有使用过URL的[Parameters] 部分。



让我们稍微修改一下这个例子，以便可以使用URL传递一些参数信息给控制器类（例如，*/HelloWorld/Welcome?name=Scott&numtimes=4*）。改变您的Welcome 方法来包含两个参数，如下所示。需要注意的是，示例代码使用了C＃语言的可选参数功能，numTimes参数在不传值时，默认值为1。

```
public string Welcome(string name, int numTimes = 1) { return HttpUtility.HtmlEncode("Hel
}
```

安全注意事项: 上面的代码使用了 HttpServerUtility.HtmlEncode 来保护应用从malacious输入的(也就是JavaScript). 有关详细信息，请参阅How to: Protect Against Script Exploits in a Web Application by Applying HTML Encoding to Strings.

运行您的应用程序并浏览此URL（*http://localhost:xxxx/HelloWorld/Welcome?name=Scott&numtimes=4__*）。你可以对参数name 和numtimes 尝试不同的值。 ASP.NET MVC model binding system 会自动将地址栏中URL里的 query string映射到您方法中的参数。



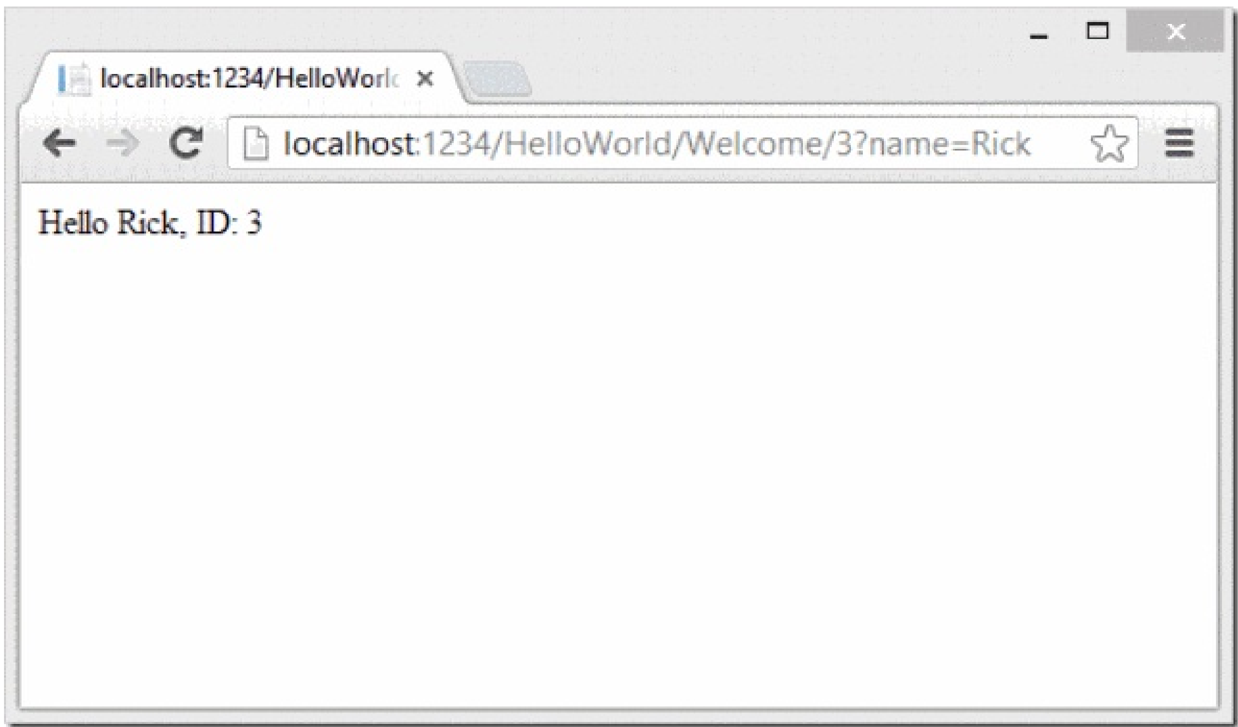
上面的例子，没有用到URL段参数的部分(Parameters)。 通过query strings传递name 和numTimes的参数.

用下面的代码替换“Welcome”的方法：

```
public string Welcome(string name, int ID = 1)

{ return HttpUtility.HtmlEncode("Hello " + name + ", ID: " + ID);
}
```

运行应用程序并输入以下网址URL: *http://localhost:xxx/HelloWorld/Welcome/3?name=Rick*



这次URL第三部分的参数匹配了参数ID。

通过查看下面的RegisterRoutes路由规则函数：

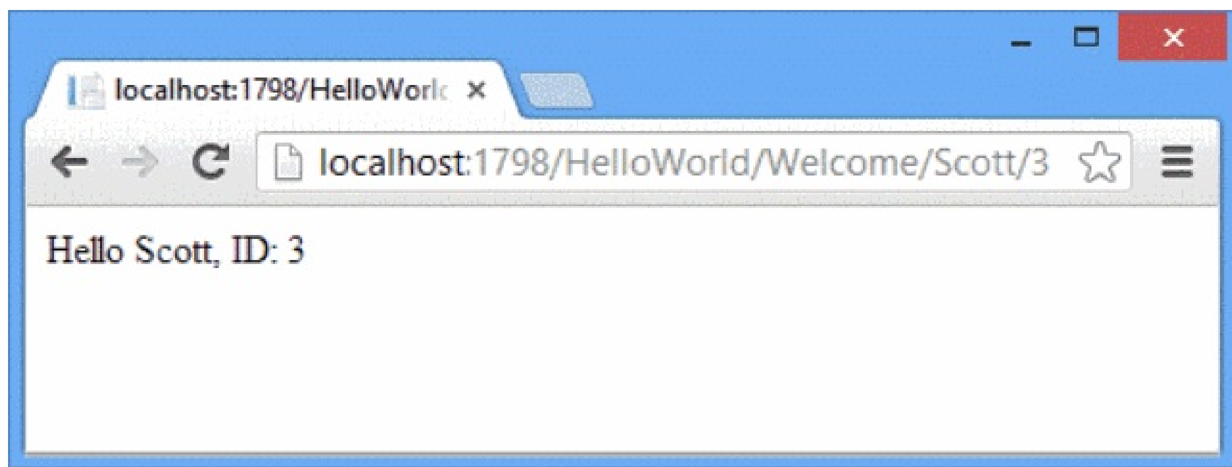
```
public static void RegisterRoutes(RouteCollection routes)

{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional
    );
}
```

在ASP.NET MVC应用程序，通过参数传递路由数据是为更典型的应用(如同上面用query string传递 ID参数)。您还可以增加一条路由来传递name 和numtimes， 在路由数据在URL中的参数。在*App_Start\RouteConfig.cs* file文件中，添加“Hello”的的路由：

```
public class RouteConfig{ public static void RegisterRoutes(RouteCollection routes)
   {
      routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
      routes.MapRoute(
          name: "Default",
          url: "{controller}/{action}/{id}",
          defaults: new { controller = "Home", action = "Index", id = UrlParameter.Option
      );
      routes.MapRoute(
           name: "Hello",
           url: "{controller}/{action}/{name}/{id}" );
   }
}
```

运用应用程序，在浏览器输入：/localhost:XXX/HelloWorld/Welcome/Scott/3.



对于众多MVC应用程序的缺省默认的路由可以正常工作。稍后您将学习本教程中通过使用模型绑定的数据，你就不必修改缺省的路由。

在上面的例子中，控制器一直在做着MVC中“VC”部分的职能：也就是视图和控制器的工作。该控制器直接返回HTML内容。通常情况下，您不会让控制器直接返回HTML，因为这样代码会变得非常的繁琐。相反，我们通常会使用一个单独的视图模板文件来帮助生成返回的HTML。让我们来看看下面我们如何能做到这一点吧。

ASP.NET MVC 5 - 视图

在本节中，你要去修改HelloWorldController类，使用视图模板文件，在干净利索地封装的过程中：客户端浏览器生成HTML。

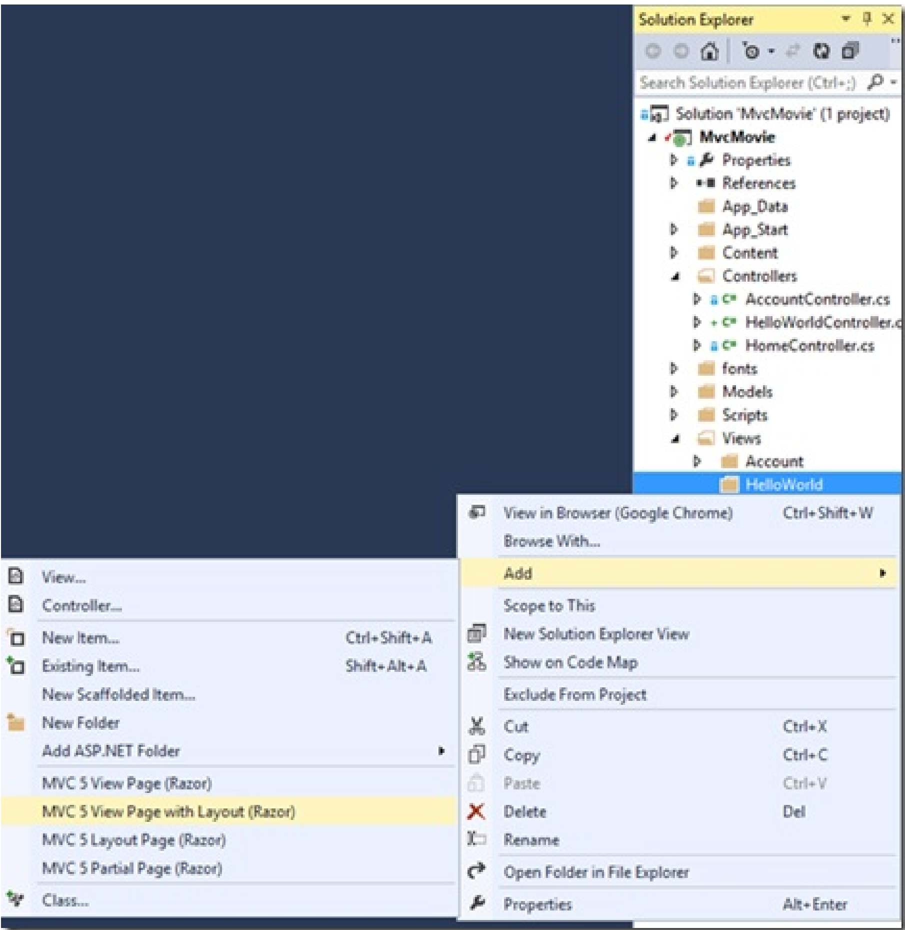
您将创建一个视图模板文件，其中使用了ASP.NET MVC 3所引入的Razor视图引擎（Razor view engine）。Razor视图模板文件使用.cshtml文件扩展名，并提供了一个优雅的方式来使用C＃语言创建所要输出的HTML。用Razor编写一个视图模板文件时，将所需的字符和键盘敲击数量降到了最低，并实现了快速，流畅的编码工作流程。

当前在控制器类中的Index方法返回了一个硬编码的字符串。更改Index方法返回一个View对象，如下面的示例代码：

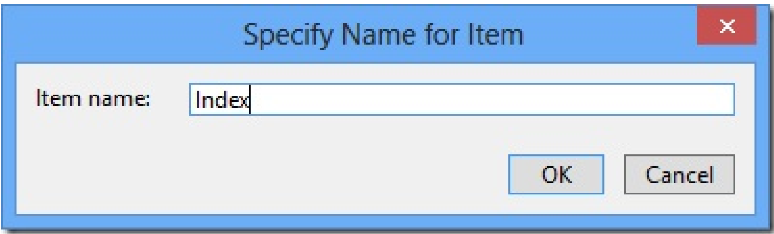
```
public ActionResult Index()
{ return View();
}
```

上面的Index方法使用一个视图模板来生成一个HTML返回给浏览器。控制器的方法（也被称为action method(操作方法)），如上面的Index方法，一般返回一个ActionResult（或从ActionResult所继承的类型），而不是原始的类型，如字符串。

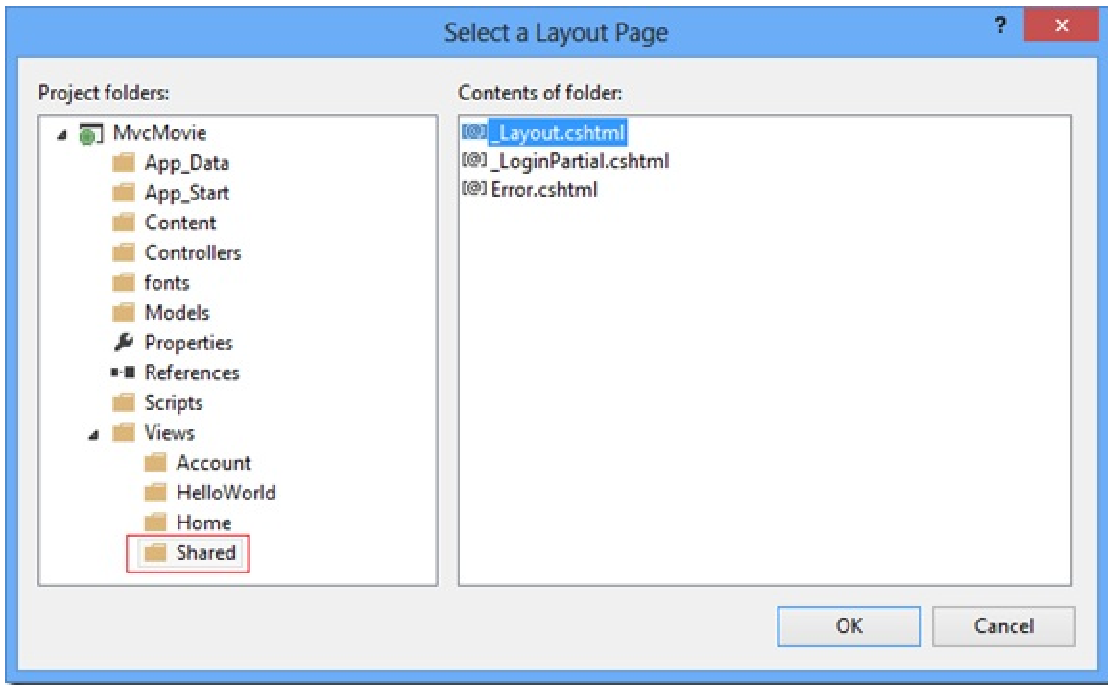
在该项目中，您可以使用的Index方法来添加一个视图模板。要做到这一点，在*Views\HelloWorld* 文件夹上，单击鼠标右键，然后单击“ 添加“，选择“**MVC 5 View Page with (Layout Razor) “**。



在"指定项名称**(Specify Name for Item)**"对话框，输入"Index "，然后单击"确定"。

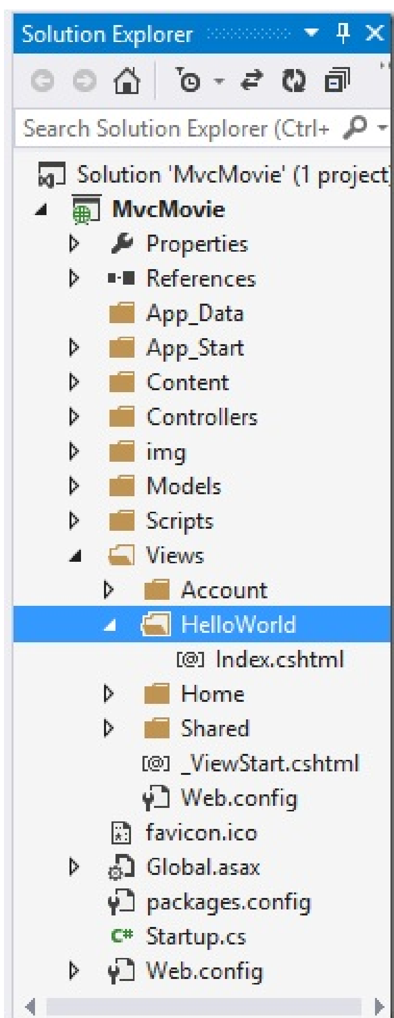


在"选择布局页**(Select a Layout Page)"对话框中，接受缺省"**_Layout.cshtml"，并单击"确定"。



在上面的对话框中，左窗格中选择的是“ **Views\Shared**”共享文件夹布局。如果你在另一个文件夹中有一个自定义布局，你也可以选择它。稍后在本教程中，我们会谈论的布局文件。

您可以在解决方案资源管理器中看到MvcMovie\HelloWorld文件夹和已被创建的MvcMovie\View\HelloWorld\Index.cshtml文件：

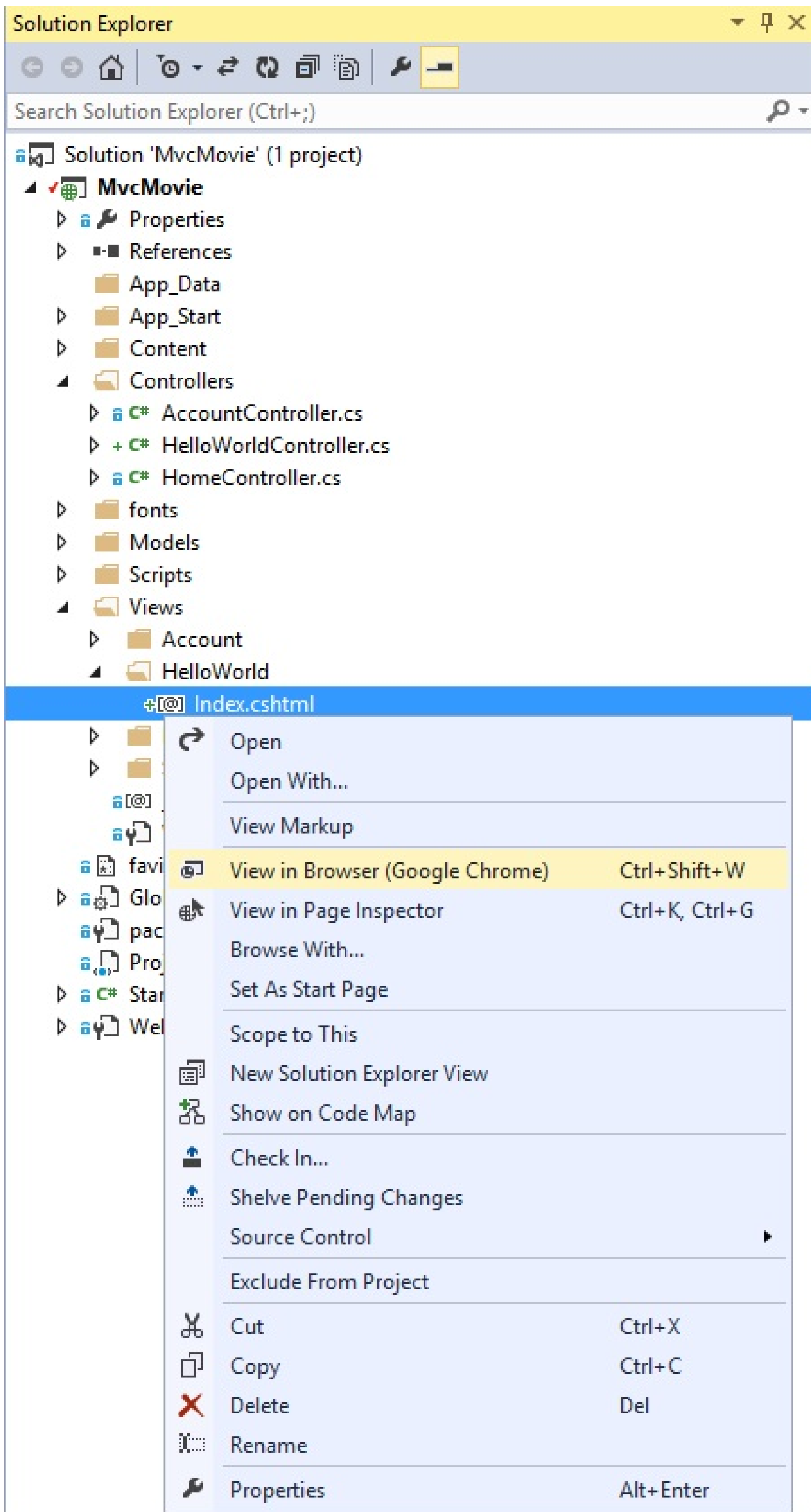


添加下面的高亮标记代码。

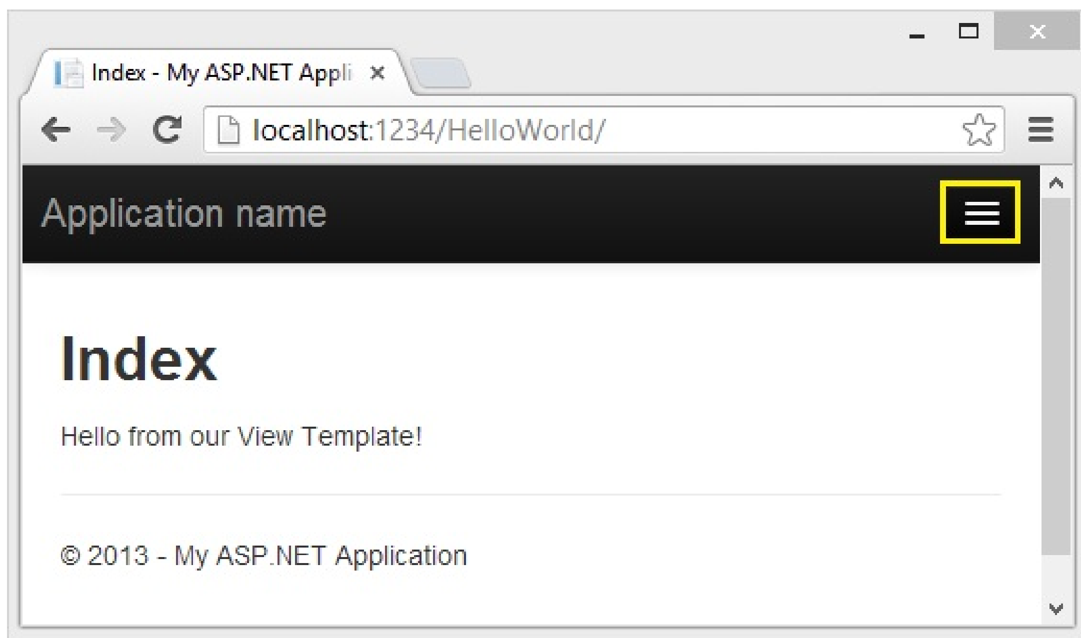
```
@{
Layout = "~/Views/Shared/_Layout.cshtml";
}
@{
ViewBag.Title = "Index";
} <h2>Index</h2>
<p>Hello from our View Template!</p>
```

在解决方案资源管理器，找到*Index.cshtml*文件，右键单击并选择“在浏览器中查看”。

页面检查器教程中会有更多的信息介绍这个工具。



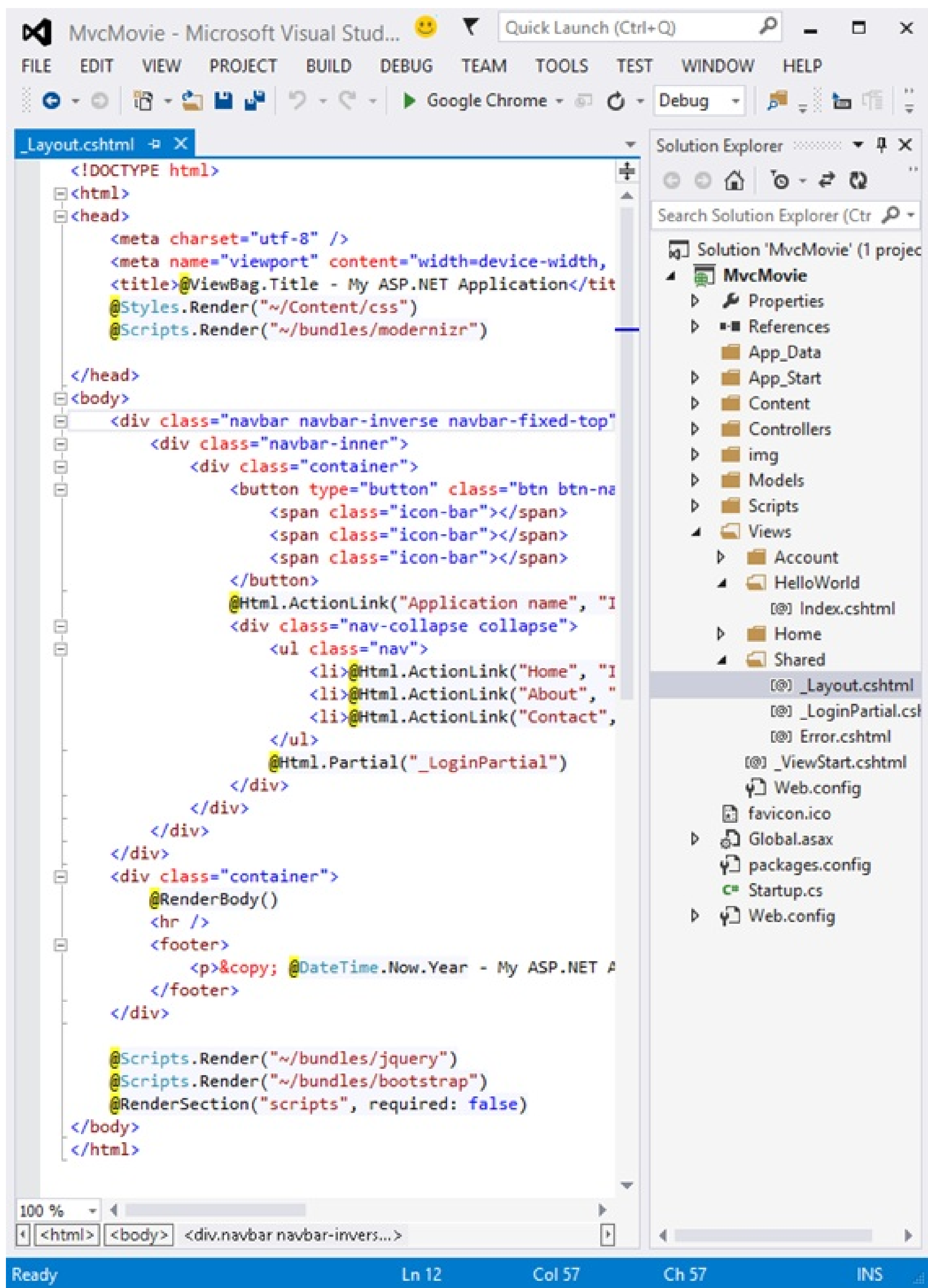
同时，运行应用程序并在浏览器中浏览：HelloWorld控制器（*http://localhost:xxxx/HelloWorld*“）。在您控制器的Index方法中并没有做太多的工作，它只是执行了return View()，这个方法指定使用一个视图模板文件来Render返回给浏览器的HTML。因为您没有明确指定使用那个视图模板文件，ASP.NET MVC会默认使用\Views\HelloWorld文件夹下的Index.cshtml视图文件。下图显示了在视图文件中硬编码的字符串 "Hello from our View Template!"



看起来很不错吧。但是，请注意，浏览器的标题栏会显示为"Index- My ASP.NET Appli" 并且在页面顶部的大链接会显示为 "Application name.”。 取决于浏览器窗口的大小，您可能需要在右上角，单击“三条杠”，首页（Home），简介（About）联系（Contact），注册（Register）和登录（Log in）的链接。

修改视图和布局页

首先，您想要修改在页面顶部的链接 "Application name"。这段文字是每个页面的公用文字，即使这段文字出现在每个页面上，但是实际上它仅保存在工程里的一个地方。在解决方案资源管理器里找到*/Views/Shared*文件夹，打开*_Layout.cshtml*文件。此文件被称为布局页面（Layout page），并且其它所有的子页面，都共享使用这个布局页面。



布局模版允许您在一个位置放置占位所需的 HTML 容器，然后将其应用到您网站中所有的网页布局。 查找 @RenderBody(). 您所创建的所有视图页面都被"包装" 在布局页面中来显示，RenderBody只是个占位符。例如，如果您点击“关于(About)”链接，*Views\Home\About.cshtml* 视图会在RenderBody方法内进行Render。

在布局模板页面内修改ActionLink.aspx)内容, 把网站标题从 " Application name " 修改为 "MVC Movie”，并修改控制器参数从Home为Movies.

完整的布局文件如下所示：

```
<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - Movie App</title> @Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr") </head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-c

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button> @Html.ActionLink("MVC Movie", "Index", "Movies", null, new { @class = "navbar-b

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul> @Html.Partial("_LoginPartial") </div>

</div>

</div>

<div class="container body-content"> @RenderBody() <hr />

<footer>

<p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>

</footer>

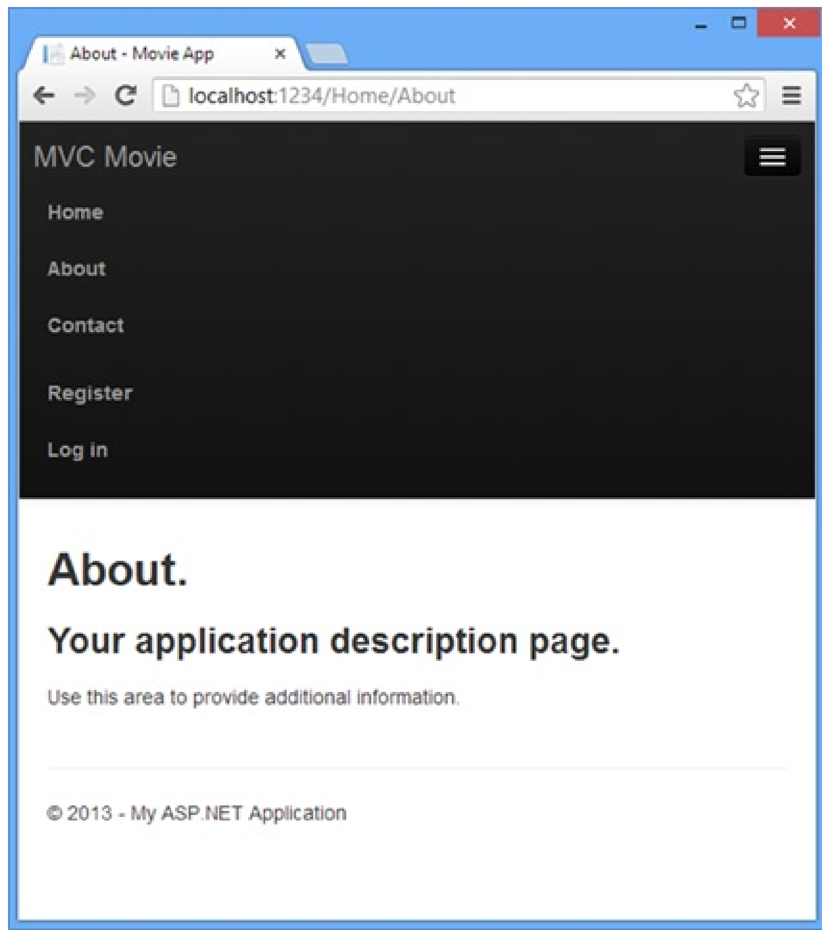
</div> @Scripts.Render("~/bundles/jquery")

@Scripts.Render("~/bundles/bootstrap")

@RenderSection("scripts", required: false) </body>

</html>
```

运行应用程序，您会看到 "MVC Movie "。 单击“ 关于**(About)**“链接，您可以看到该页面也会显示为"MVC Movie "。 我们可以在布局模版里再修改一次，使得网站里所有网页的标题都同时被修改掉。



打开创建的 *Views\HelloWorld\Index.cshtml*文件，可以找到如下代码:

```
@{
Layout = "~/Views/Shared/_Layout.cshtml";
}
```

上面的 Razor 代码，显示的设置了布局页面。打开 *Views_ViewStart.cshtml* 文件, 它也有同样的Razor 标记代码。*Views_ViewStart.cshtml* 文件定义我们使用到的所有视图的通用布局，故你也可在*Views\HelloWorld\Index.cshtml*文件里面，注释或删除这些代码。

```
@*@{
Layout = "~/Views/Shared/_Layout.cshtml";
}*@
@{
ViewBag.Title = "Index";
} <h2>Index</h2>
<p>Hello from our View Template!</p>
```

你可以使用Layout属性设置一个不同的布局页面，或者设置为null指明不使用布局文件

现在，让我们来修改Index视图：

打开MvcMovie\Views\HelloWorld\Index.cshtml文件，有两个地方需要进行修改：

· 浏览器上的标题文字

· 其次，二级标题文字 (`<h2>` 元素)。

让它们稍有不同，这样就可以看出到底程序里那部分的代码被修改了。

```
@{
ViewBag.Title = "Movie List";
} <h2>My Movie List</h2>
<p>Hello from our View Template!</p>
```

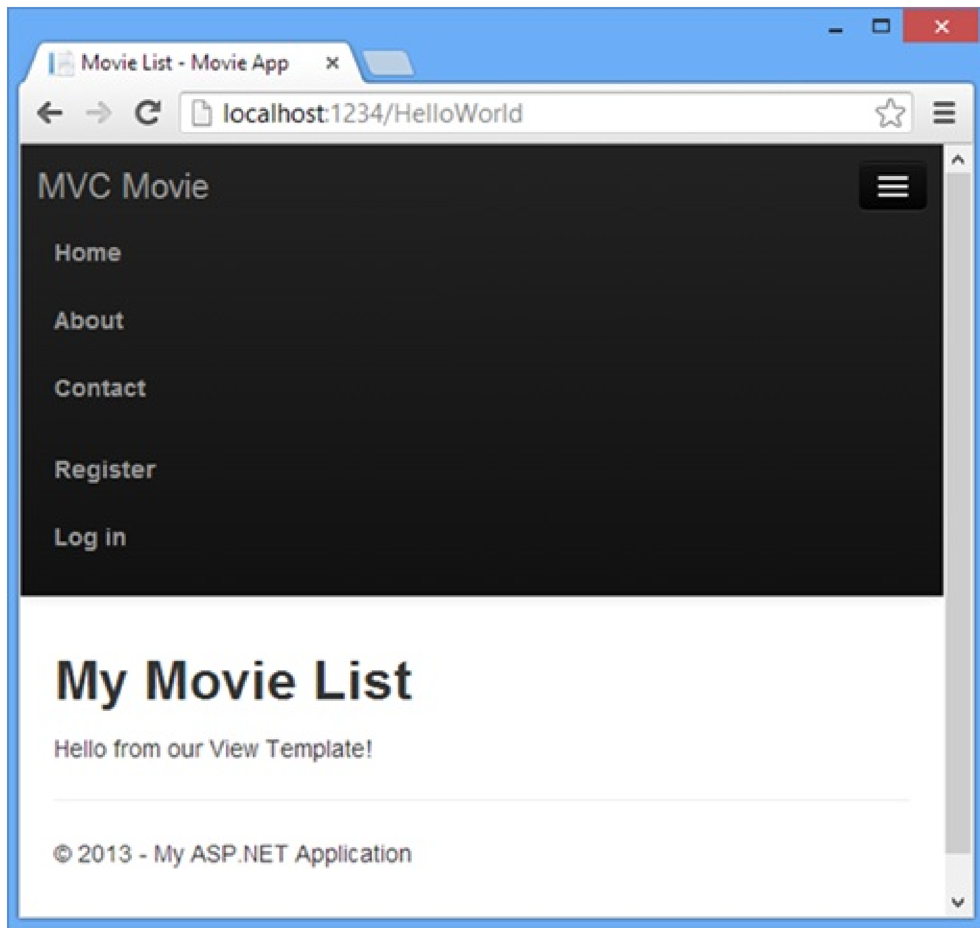
如果要指定HTML的title元素，上面的代码设置了 `ViewBag` 对象 （在*Index.cshtml*视图模板中） 的 `Title` 属性。如果您回去看看布局模板的源代码，您会发现该模板会输出此值倒 `<title>` 元素中，从而作为我们之前修改过的 HTML `<head>` 里的一部分。

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>@ViewBag.Title - Movie App</title> @Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr") </head>
```

使用此 `ViewBag` 方法，您可以轻松地从视图模板传递其它参数给布局模板页面。

运行应用程序，浏览http://localhost:xx/HelloWorld。 浏览器的标题、 主标题和二级标题都已经被修改了。（如果您在浏览器中看不到修改，有可能是页面被缓存了。按 Ctrl + F5 强制浏览器重新请求并加载服务器返回的HTML） 在*Index.cshtml*视图模版中设置的 `ViewBag.Title` 输出了浏览器的标题，附加的"- Movie App"是在布局模板文件中添加的。

此外还要注意*Index.cshtml*视图模板中的内容是如何合并到*_Layout.cshtml*模板，从而形成一个完整的HTML返回到客户端浏览器的。使用布局模板页面，可以很容易进行一个修改并应用到所有页面。



我们这一点（在本例中的"Hello from our View Template!"字符串） 的"数据" 只是一段硬编码。这个MVC 应用程序有了一个"V"（视图）， 也有了一个"C"（控制器）， 但还没有"M"(模型)。不过稍后，我们将介绍如何创建一个数据库并检索数据模型。

ASP.NET MVC 5 - 将数据从控制器传递给视图

在我们讨论数据库和数据模型之前，让我们先讨论一下如何将数据从控制器传递给视图。控制器类将响应请求来的URL。控制器类是给您写代码来处理传入请求的地方，并从数据库中检索数据，并最终决定什么类型的返回结果会发送回浏览器。视图模板可以被控制器用来产生格式化过的HTML从而返回给浏览器。

控制器负责给任何数据或者对象提供一个必需的视图模板，用这个视图模板来Render返回给浏览器的HTML。最佳做法是：一个视图模板应该永远不会执行业务逻辑或者直接和数据库进行交互。相应的，一个视图模板应该只和控制器所提供的数据进行交互。维持这种"隔离关系"可以帮助，保持代码的干净、测试性和更易维护。

当前， `HelloWorldController` 类中 `Welcome` 操作方法需要一个 `name` 和一个 `numTimes` 参数，然后直接输出给浏览器。相比只返回一个字符串，让我们来改变控制器，来使用视图模板吧。视图模板将生成动态的HTML，这意味着您需要通过适当的方式把数据从控制器传递给视图，从而才能生成动态的HTML。您可以把视图模板需要的动态数据（参数）在控制器中放入到一个 `ViewBag` 对象中，然后视图模板可以访问这个对象。

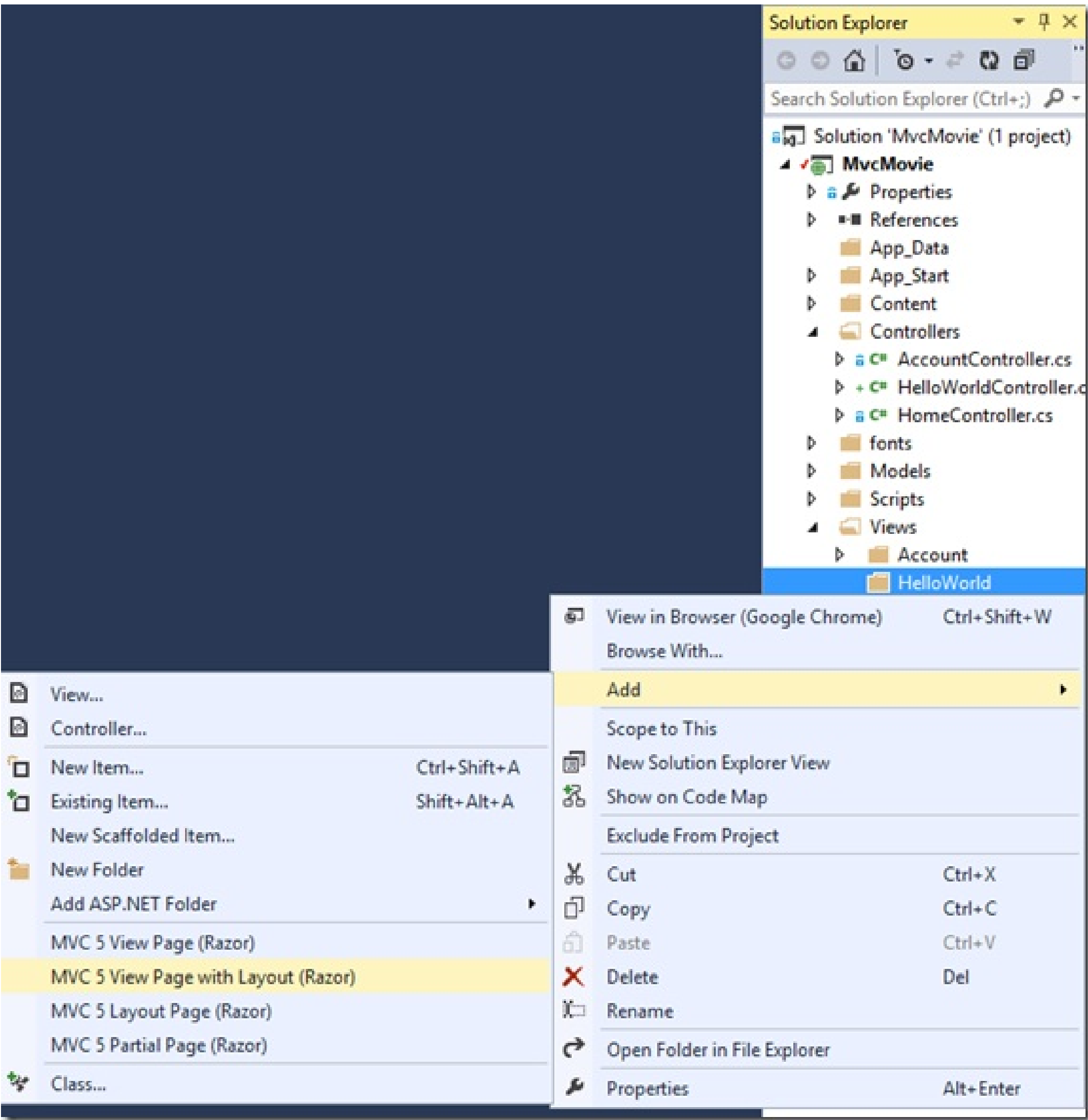
打开*HelloWorldController.cs*文件，更改 `Welcome` 方法，将 `Message` 和 `NumTimes` 的值添加到ViewBag对象里。 `ViewBag` 是一个动态的对象，这意味着在您没有给 `ViewBag` 放置属性时，它没有任何属性，您可以把任何您想放置的对象放入到 `ViewBag` 对象中。 ASP.NET MVC model binding system 会自动将地址栏中URL里的 query string映射到您方法中的参数（ `name` 和 `numTimes` ）。

完整的*HelloWorldController.cs*文件如下所示：

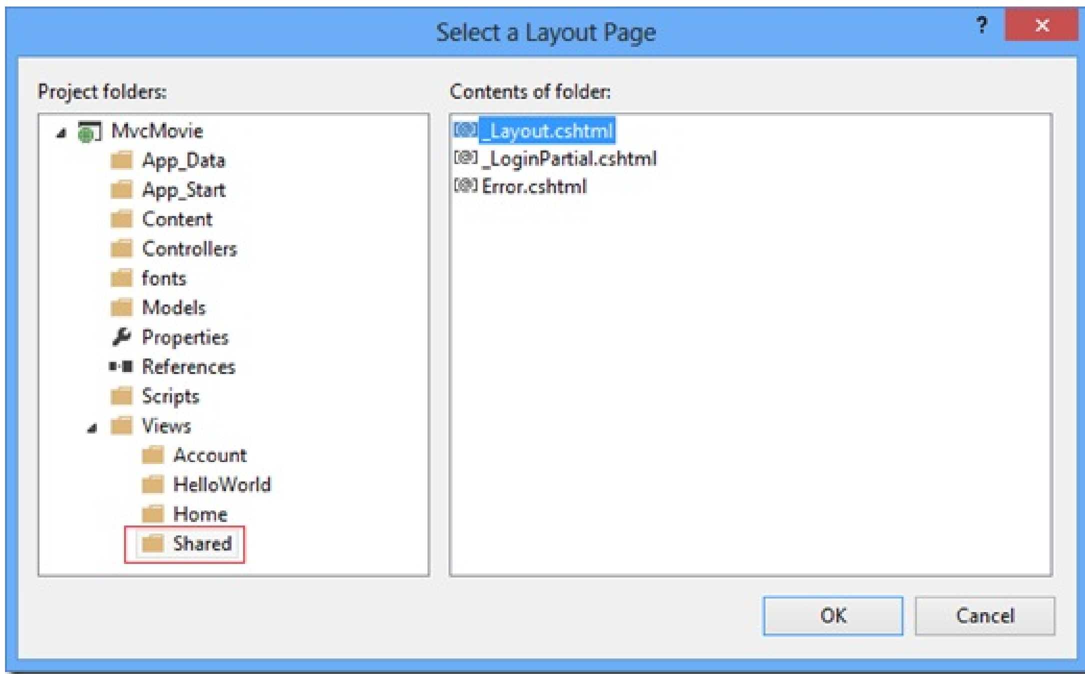
```
using System.Web; using System.Web.Mvc; namespace MvcMovie.Controllers
{ public class HelloWorldController : Controller
{ public ActionResult Index()
{ return View();
} public ActionResult Welcome(string name, int numTimes = 1)
{
ViewBag.Message = "Hello " + name;
ViewBag.NumTimes = numTimes; return View();
}
}
}
```

现在 `ViewBag` 对象包含了数据，并将自动传递给视图模板。 接下来，您需要一个欢迎视图模板 ！在生成菜单中，选择生成** MvcMovie (快捷键 Ctrl+Shift+B)** ，以确保项目编译成功。

在*Views\HelloWorld*文件夹上，右键单击”添加**(视图)”**，选择”__*MVC 5 View Page with (Layout Razor).”*



在 “指定项名称** (**Specify Name for Item**)” 对话框, 输入” Welcome”, 点击“确定****(OK)**”.
在“选择布局（**the Select a Layout Page**）”对话框， 接受缺省的”布局**_Layout.cshtml”,
并点击“确定****(OK)**”.



*MvcMovie\Views\HelloWorld\Welcome.cshtml*文件创建成功。

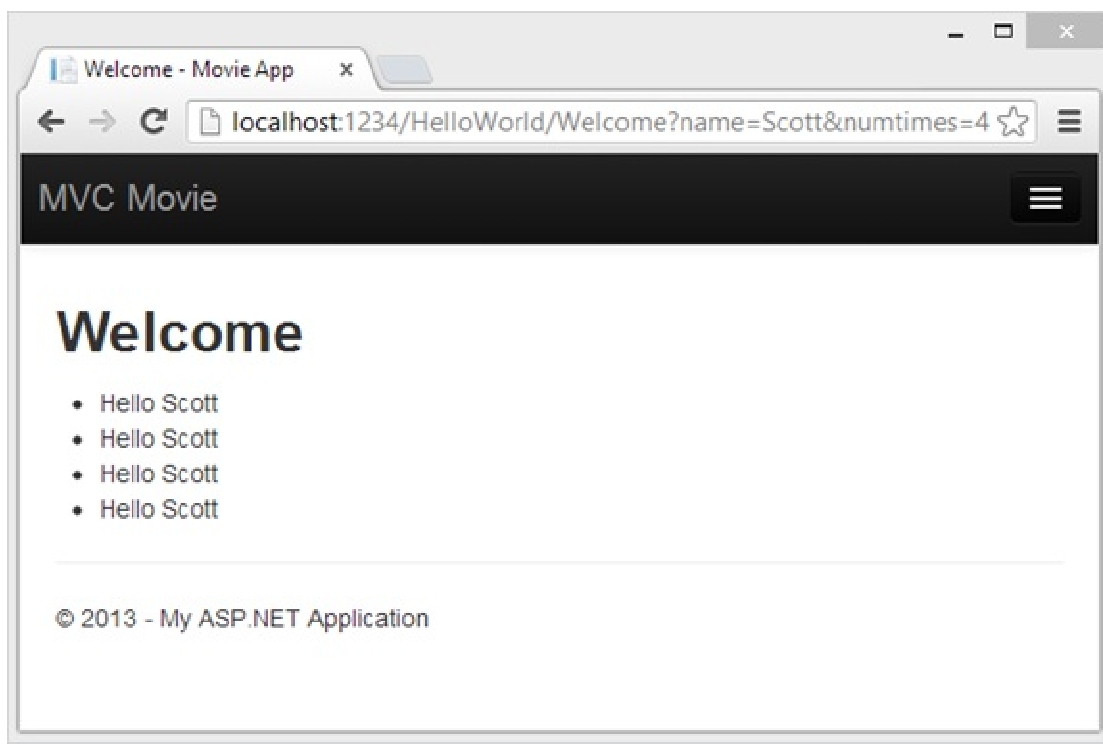
在Welcome.cshtml文件里替换标记，您将创建一个循环，循环说多次“Hello”。

下面显示了完整的*Welcome.cshtml*文件。

```
@{
ViewBag.Title = "Welcome";
} <h2>Welcome</h2>
<ul> @for (int i = 0; i < ViewBag.NumTimes; i++)
{ <li>@ViewBag.Message</li> } </ul>
```

运行应用程序，并浏览下面的 URL : *http://localhost:xx/HelloWorld/Welcome?name=Scott&numtimes=4*

现在，模型绑定(model binder) 使得数据从URL传递给控制器。控制器将数据装入到 `ViewBag` 对象中，通过该对象传递给视图。然后视图为用户生成显示所需的HTML。



在上面的示例中，我们使用了 `ViewBag` 对象把数据从控制器传递给了视图。在本系列教程后面的文章中，我们将使用视图模型来将数据从一个控制器传递到视图中。用视图模型来传递数据，这一般是首选的办法。Blog Dynamic V Strongly Typed Views 有更加详细的介绍。

到这里，这是一种"M"模型，但不是数据库的那种“M”模型。让我们来创建一个电影数据库吧。

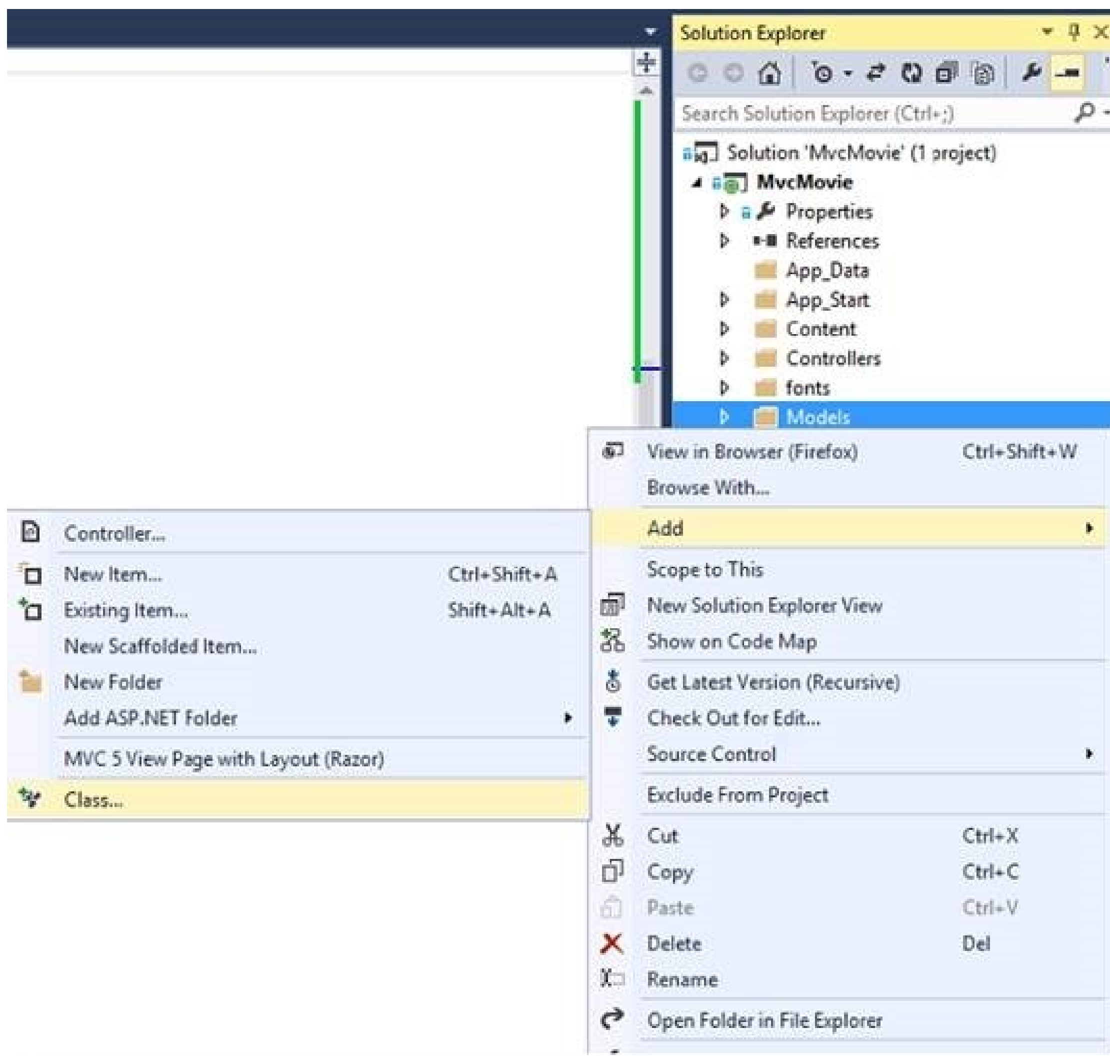
ASP.NET MVC 5 - 添加一个模型

在本节中，您将添加一些类，这些类用于管理数据库中的电影。这些类是ASP.NET MVC 应用程序中的"模型(Model)"。

您将使用.NET Framework 数据访问技术Entity Framework.aspx)，来定义和使用这些模型类。Entity Framework（通常称为 EF) 是支持代码优先 (Code First) 的开发模式。代码优先允许您通过编写简单的类来创建对象模型。（相对于"原始的CLR objects"，这也被称为POCO类）然后，可以从您的类创建数据库，这是一个非常干净快速的开发工作流程。假如你必须首先创建数据库，你依旧也可遵循这个教程，以了解MVC和EF应用程序开发。然后，您可以遵循Tom Fizmakens ASP.NET的Scaffolding教程，其涵盖了首先创建数据库的方法。

添加模型类

在解决方案资源管理器中，右键单击模型文件夹，选择添加，然后选择类.



输入*Class*名 "Movie"。

将下列五个属性添加到 `Movie` 类：

```
using System; namespace MvcMovie.Models
{ public class Movie
    { public int ID { get; set; } public string Title { get; set; } public DateTime Relea
    }
}
```

我们将使用 `Movie` 类来表示数据库中的电影。 `Movie` 对象的每个实例将对应数据库表的一行， `Movie` 类的每个属性将对应表的一列。

在同一文件中，添加下面的 `MovieDBContext` 类：

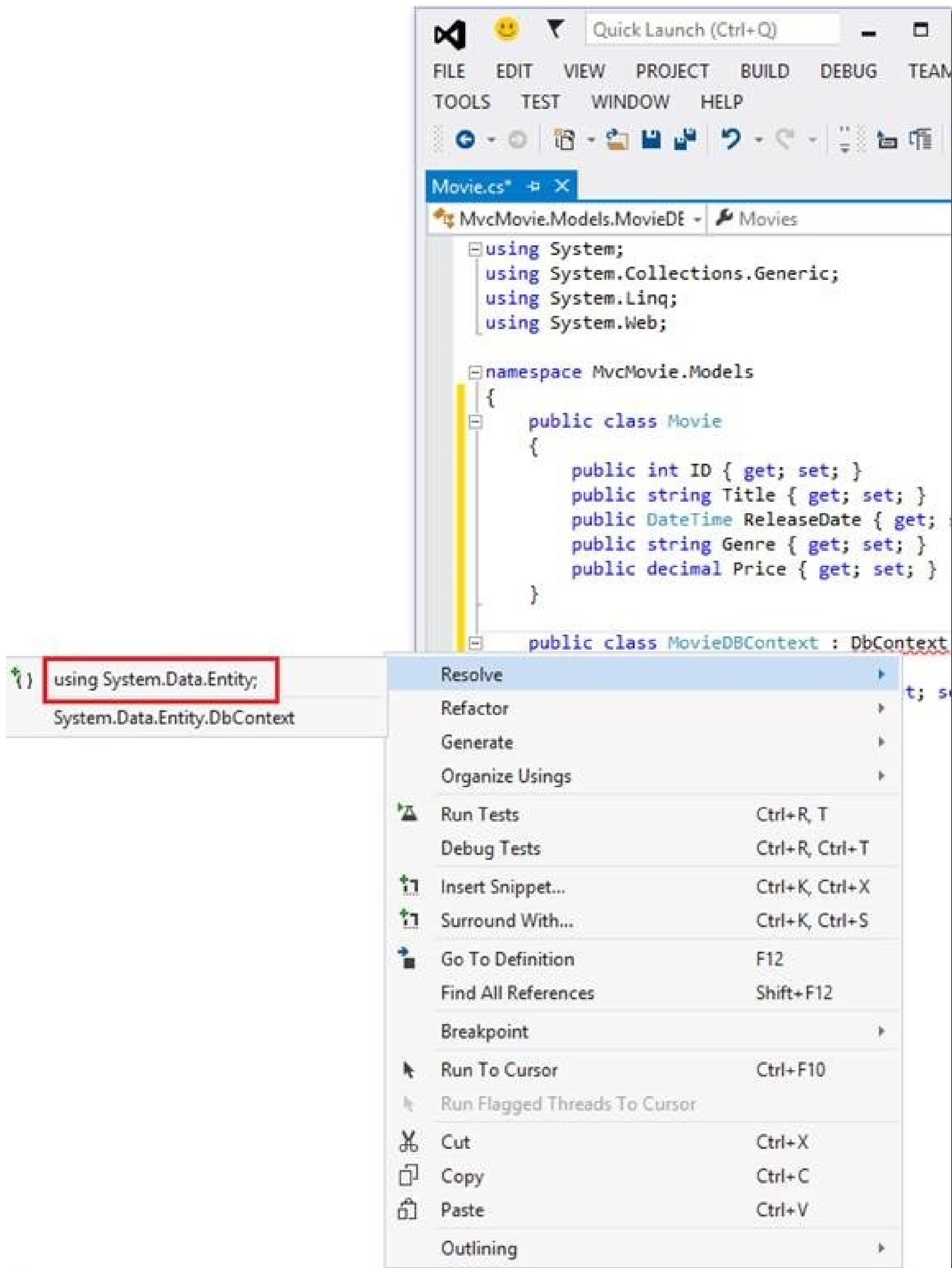
```
using System; using System.Data.Entity; namespace MvcMovie.Models
{ public class Movie
    { public int ID { get; set; } public string Title { get; set; } public DateTime Relea
    } public class MovieDBContext : DbContext
    { public DbSet<Movie> Movies { get; set; }
    }
}
```

`MovieDBContext` 类代表Entity Framework的电影数据库类，这个类负责在数据库中获取，存储，更新，处理 `Movie` 类的实例。 `MovieDBContext` 继承自Entity Framework的 DbContext.aspx)基类。

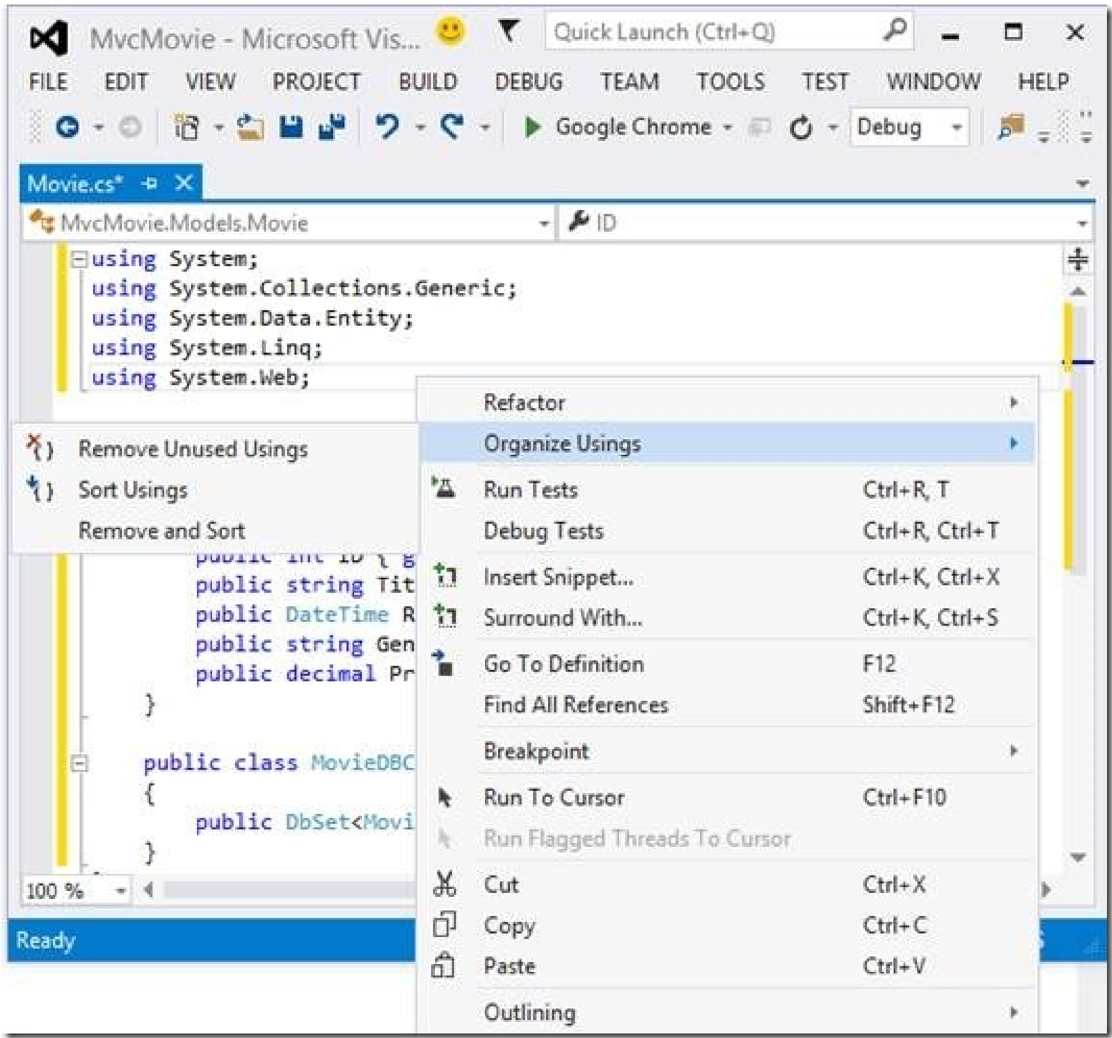
为了能够引用 `DbContext` 和 `DbSet` ，您需要在文件的顶部添加以下 `using` 语句：

```
using System.Data.Entity;
```

为此，您可以通过手动添加using语句，或者您可以右键单击红色的波浪线，“解析（Resolve）”，然后单击“**using System.Data.Entity**。



注意：一些不用的using语句已经被删除了--通过在文件中右键单击，选择“组织**Using”，然后单击”移除未使用的**using”。



到此为止，我们增加了一个模型（MVC**中的**M）。 在下一节中，您将使用的数据库连接字符串。

ASP.NET MVC 5 - 创建连接字符串(Connection String)并使用SQL Server LocalDB

您创建的MovieDBContext类负责处理连接到数据库，并将Movie对象映射到数据库记录的任务中。你可能会问一个问题，如何指定它将连接到数据库？实际上，确实没有指定要使用的数据库，Entity Framework将预设值使用的LocalDB。在本节中，我们将显式地在Web.config文件中，添加应用程序的连接字符串(connection string)。

SQL Server Express LocalDB

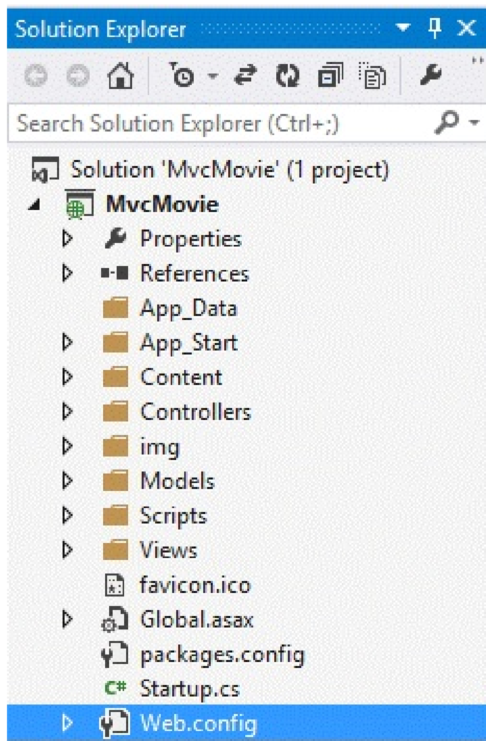
LocalDB的是一个SQL Server Express轻量级版本的数据库引擎。它在用户模式下启动、执行。LocalDB的运行在一个特殊的SQL Server Express的执行模式，所以允许您使用MDF文件数据库。通常情况下，LocalDB的数据库文件都保存在web项目的App_Data文件夹下面。

注意：在生产环境的Web应用程序中，我们不推荐您使用SQL Server Express。尤其，LocalDB不应该被用于Web应用程序的生产环境，因为它设计之初不要求使用**IIS**。然而，LocalDB的数据库能够很容易地迁移到SQL Server或SQL Azure中。

备注：在 Visual Studio 2013 (Visual Studio 2012), LocalDB 默认会被安装.

默认的，Entity Framework的看起来命名为为对象上下文类（如本项目MovieDBContext）的相同的一个连接字符串。有关详细信息，请参见SQL Server Connection Strings for ASP.NET Web Applications。

打开应用程序根目录的*Web.config*文件。（不是*View*文件夹下的*Web.config*文件。）打开红色高亮标记的*Web.config*文件。

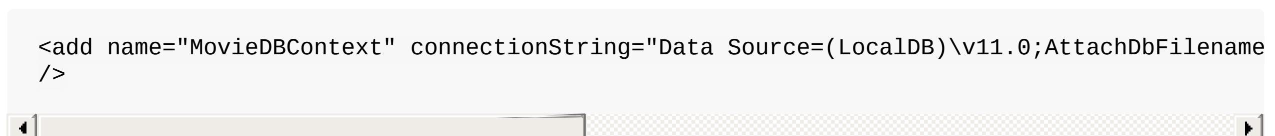


找到 <connectionStrings>：



```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visi
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile
  </configSections>
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="PreserveLoginUrl" value="true" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
  <system.web>
```

在*Web.config*文件中的 `<connectionStrings>` 内添加下面的连接字符串。



```
<add name="MovieDBContext" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename
/>
```

下面的例子里显示了部分*Web.config*文件中所新添加的连接字符串：

```
<connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;AttachDbF
    <add name="MovieDBContext" connectionString="Data Source=(LocalDB)\v11.0;AttachDbFile
/>
```

这两个连接字符串非常相似。第一个连接字符串命名为DefaultConnection的，被用于控制可以访问应用程序的成员鉴权数据库。您已添加的连接字符串 (connection string) 显示位于AppData文件夹中的 一个*Movie.mdf*文件，数据库命名为_*Movie.mdf*。 在本教程中，我们将不使用会员数据库有关会员，认证和安全性的更多信息，请参阅教程: Deploy a Secure ASP.NET MVC app with Membership, OAuth, and SQL Database to a Windows Azure Web Site。

连接字符串（connection string）的名称必须匹配DbContext.aspx)类的名称。

```
using System; using System.Data.Entity; namespace MvcMovie.Models
{ public class Movie
    { public int ID { get; set; } public string Title { get; set; } public DateTime Relea
    } public class MovieDBContext : DbContext
    { public DbSet<Movie> Movies { get; set; }
    }
}
```

实际上, 您并不需要新增MovieDBContext连接字符串。 如果没有指定一个连接字符串，Entity Framework将会在用户目录中创建一个LocalDB数据库的DbContext.aspx)类的（如本例中MvcMovie.Models.MovieDBContext）。您也数据库命名为任何你喜欢的东西，只要它具有*.MDF*的后缀。例如，我们可以命名数据库*MyFilms.mdf*。

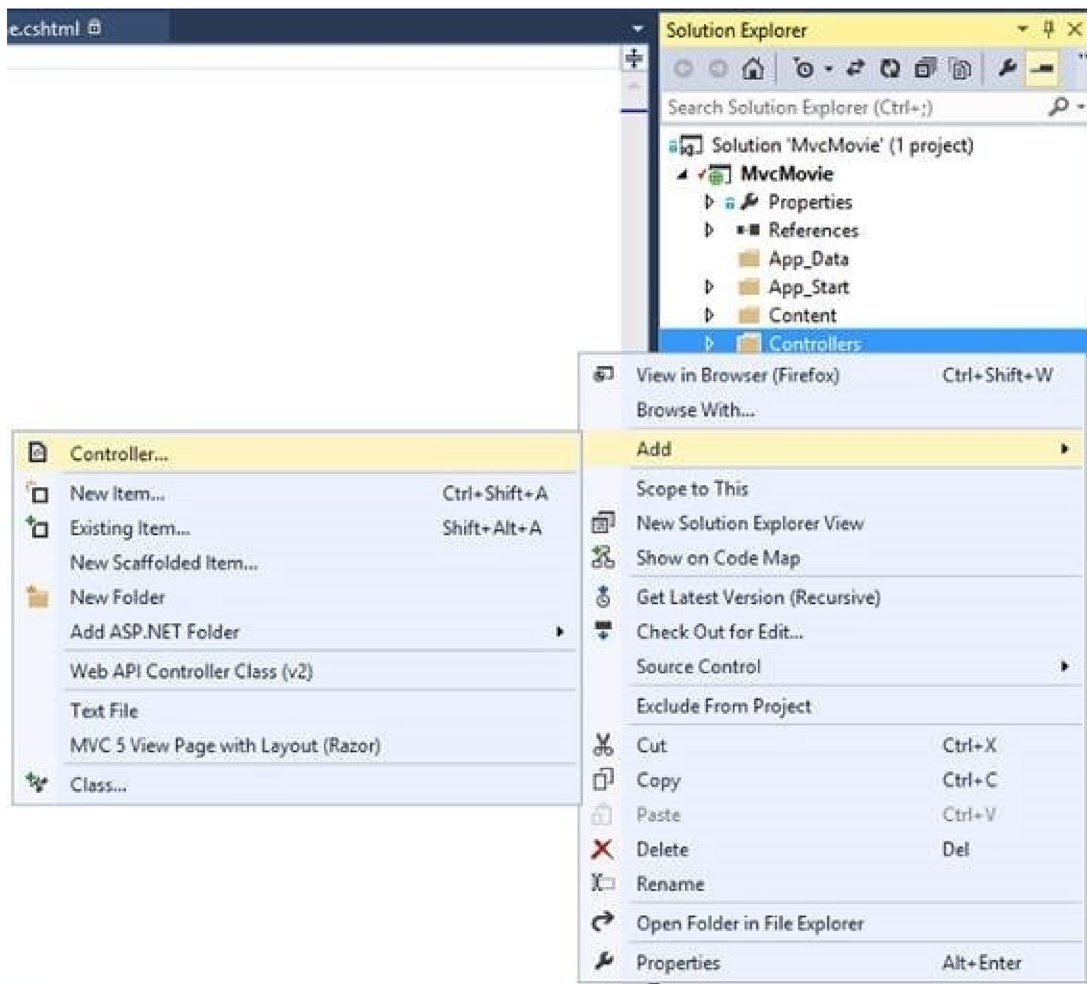
接下来，您将创建一个新的MoviesController类，您可以用它来展示电影数据，并允许用户创建新的影片列表。

ASP.NET MVC 5 -从控制器访问数据模型

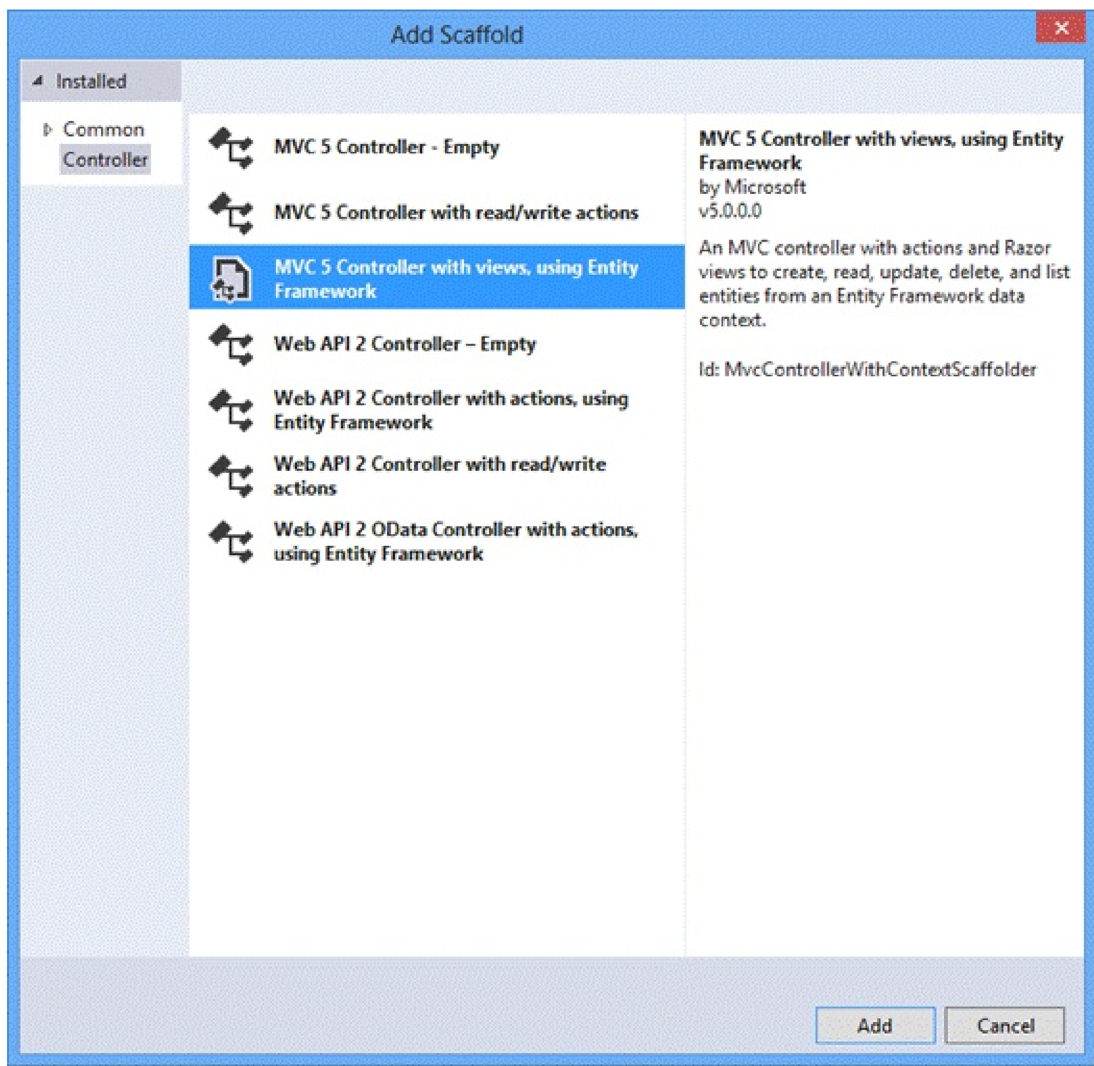
在本节中，您将创建一个新的 `MoviesController` 类，并在这个Controller类里编写代码来取得电影数据，并使用视图模板将数据展示在浏览器里。

在开始下一步前，先Build一下应用程序(生成应用程序)(确保应用程序编译没有问题)

在解决方案上，用鼠标右键单击Controller文件夹，点击新增，再选择**Controller**。



在**Scaffold**新增对话框，选择MVC 5 Controller with views, using Entity Framework**, 点击新增。

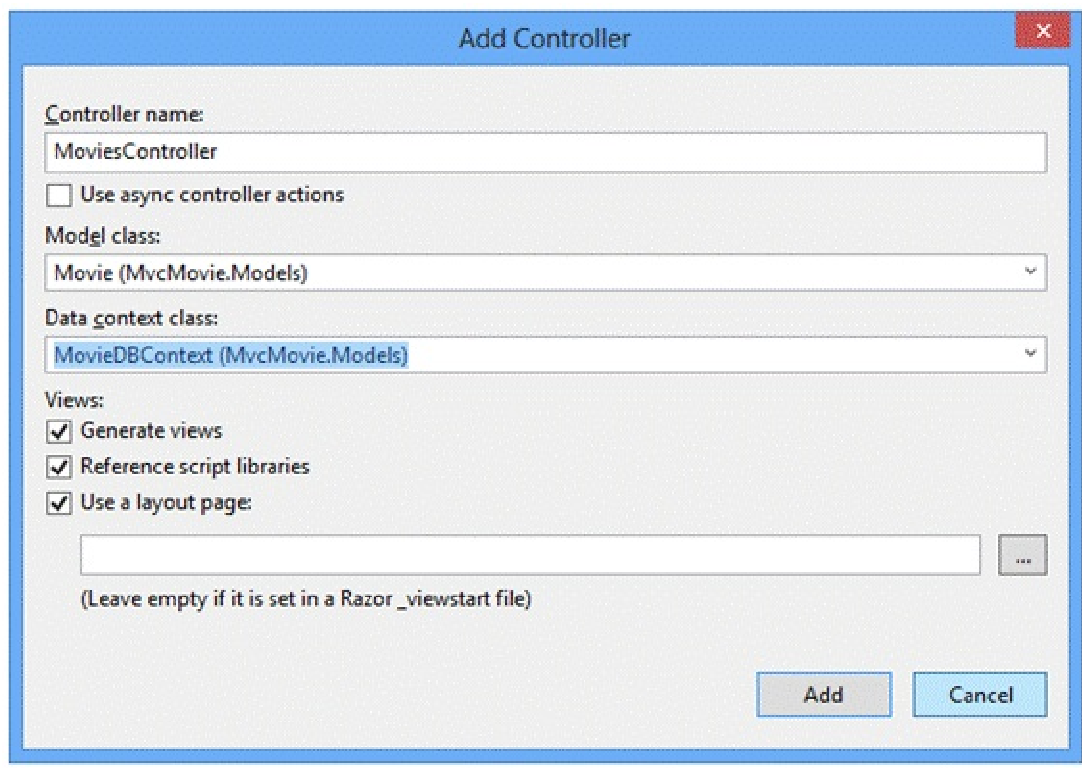


· 控制器(Controller)名称输入： **MoviesController**.

· 模型类（Model class）选择： **Movie (MvcMovie.Models)** .

· 数据上下文类(Data context class)选择：**MovieDBContext (MvcMovie.Models)**

下图显示了完成的对话框。



单击添加（如果你得到一个错误，则很可能增加控制器前，没有生成该应用程序）。Visual Studio Express 会创建以下文件和文件夹：

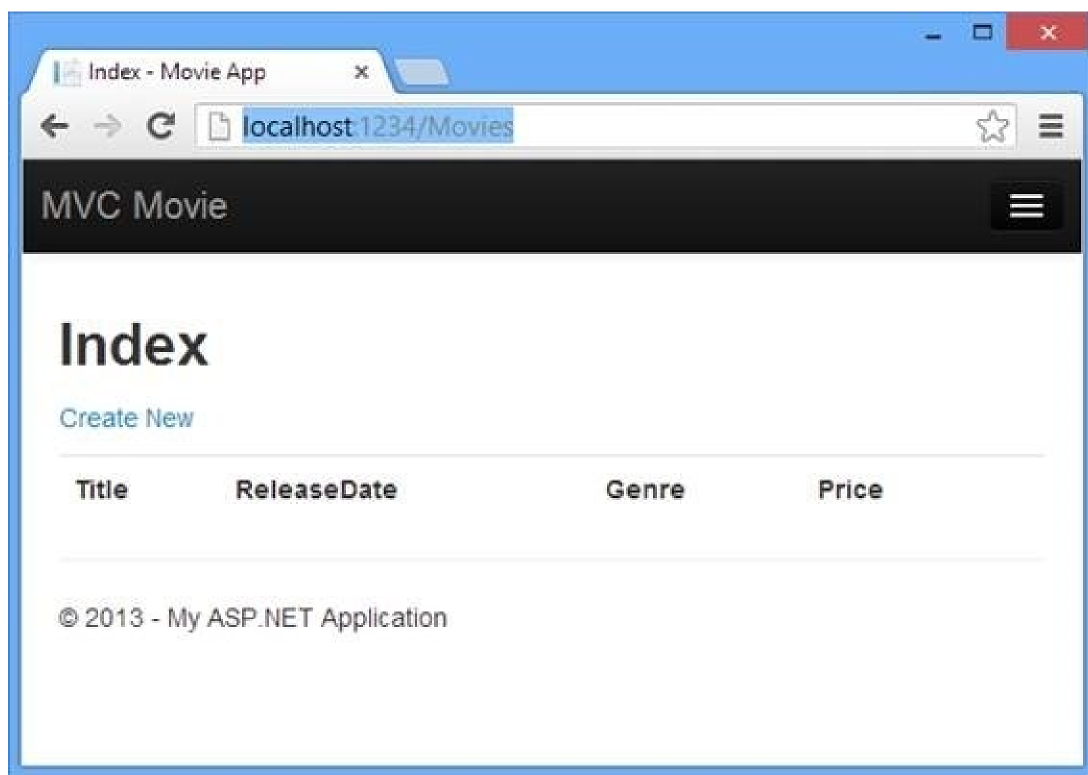
· 项目控制器文件夹中的*MoviesController.cs*文件。

· 项目视图文件夹下的 Movie文件夹。

· 在新的Views\Movies文件夹中创建*Create.cshtml*、 *Delete.cshtml*、 *Details.cshtml*、 *Edit.cshtml*和*Index.cshtml* 文件。

Visual Studio自动创建 CRUD（创建、 读取、 更新和删除） 操作方法，和相关的视图文件(CRUD 自动创建的操作方法和视图文件被称为 **scaffolding**)。 现在您有了可以创建、列表、编辑和删除电影Entity 所有的Web功能了。

运行应用程序，通过将/Movies追加到浏览器地址栏 URL的后面，从而浏览Movies控制器。因为应用程序依赖于默认路由 （ *App_Start\RouteConfig.cs* 文件中的定义），浏览器请求*http://localhost:xxxxx/Movies*将被路由到 `Movies` 控制器默认的 `Index` 操作方法。换句话说，浏览器请求*http://localhost:xxxxx/Movies*等同于浏览器请求*http://localhost:xxxxx/Movies/Index*。因为您还没有添加任何内容，所以结果是一个空的电影列表。

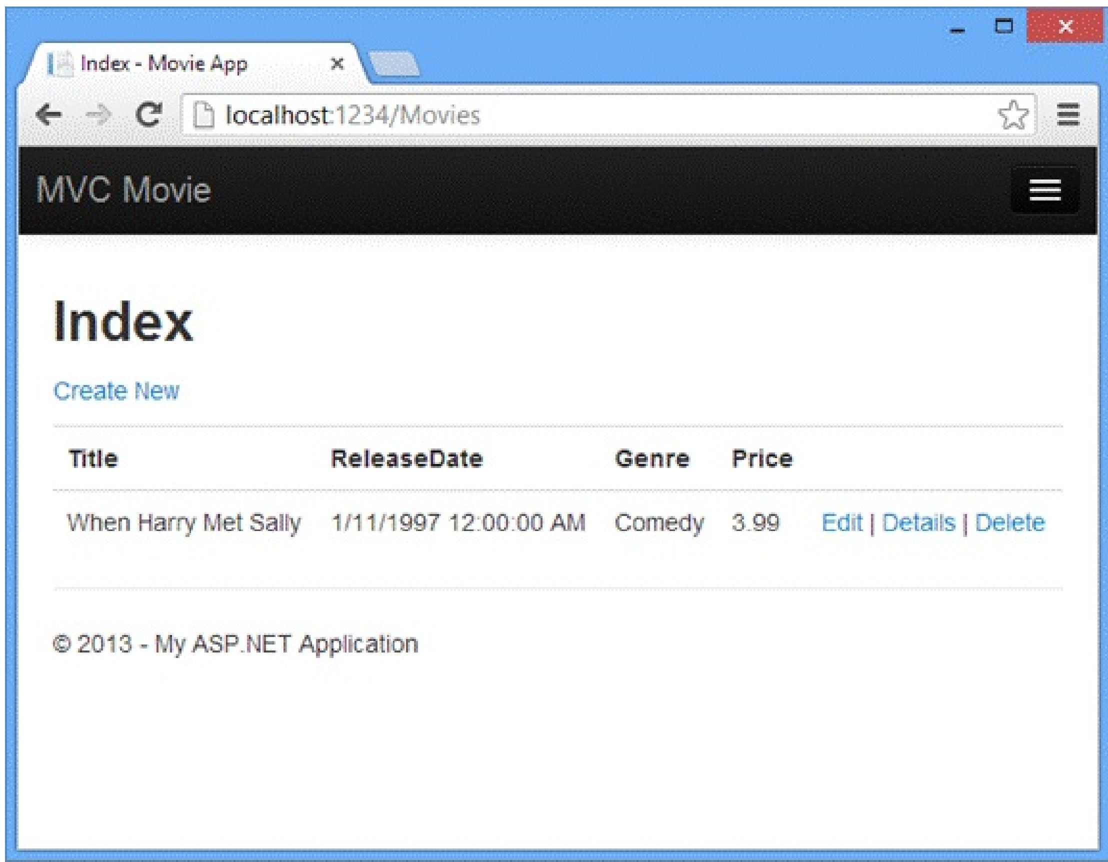


创建电影

点击**Create New**链接。输入有关电影的一些详细信息，然后单击**Create**按钮。

注意：您可能无法在“价格”字段中输入小数点或逗号。要支持非英语语言环境，小数点用逗号(","），和非美国英语的日期格式的jQuery验证，你必须包括globalize.js，和你的具体文化/globalize.cultures.js的文件（从https://github.com/jquery/globalize）和JavaScript使用Globalize.parseFloat的。在接下来的教程中，我将展示如何做到这一点。现在，只需输入整数，如10。

单击**Create**按钮将使得窗体提交至服务器，同时电影信息也会保存到数据库里，然后您会被重定向到/Movies 的URL，您可以在列表中看到刚刚创建的新电影。



创建一些更多的电影数据（movie entries）。 同时也可以尝试点击编辑、详细信息和删除功能的链接。

看一下生成的代码

打开*Controllers\MoviesController.cs*文件，并找到生成的 `Index` 方法。一部分电影控制器和 `Index` 方法如下所示。

```
public class MoviesController : Controller
{
  private MovieDBContext db = new MovieDBContext(); // GET: /Movies/

  public ActionResult Index()
  {
    return View(db.Movies.ToList());
  }
}
```

向 `Movies` 控制器请求，从而返回 `Movies` 电影数据库表中的所有记录，然后将结果传递给 `Index` 视图。下面是MoviesController类中实例化电影数据库上下文实例，如前面所述。电影数据库上下文实例可用于查询、 编辑和删除的电影。

```
private MovieDBContext db = new MovieDBContext();
```

强类型模型和 @model 关键字

在本系列前面教程中，您看到了使用 `ViewBag` 对象，从控制器传递数据或对象给视图模板。 `ViewBag` 是一个动态的对象，提供了方便的后期绑定（late-bound）方法将信息传递给视图。

MVC 还提供了传递强类型对象（strongly typed objects）到视图模板的能力。这种强类型使得更好的在编译时检查您的代码, 并在Visual Studio 编辑器中提供更加丰富的智能感知.aspx)(IntelliSense.aspx))。 当创建操作方法和视图时， Visual Studio 中的scaffolding机制（也就是通过一个强类型的模型）使用了 `MoviesController` 类和视图模板。

在*Controllers\MoviesController.cs*文件中看一下生成的 `Details` 方法。电影控制器里的 `Details` 方法如下所示。

```
public ActionResult Details(int? id)
{
  if (id == null)
  {
    return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
  }
  Movie movie = db.Movies.Find(id);
  if (movie == null)
  {
    return HttpNotFound();
  }
  return View(movie);
}
```

id参数一般是通过路由数据传递. 例如 http://localhost:1234/movies/details/1 会设置电影控制器的控制，该方法操作details并设置id为1。 你也可以通过一个查询字符串(query string) 的id如下： http://localhost:1234/movies/details?id=1

如果查找到了一个 `Movie` ， `Movie` 模型的实例会传递给Detail视图。

```
return View(movie);
```

看一下Views\Movies\Details.cshtml文件里的内容。

```
@model MvcMovie.Models.Movie @{ ViewBag.Title = "Details"; } <h2>Details</h2> <div> <h4>M

            @Html.DisplayNameFor(model => model.Title)

</dt>

         @*Markup omitted for clarity.*@

</dl> </div> <p>

    @Html.ActionLink("Edit", "Edit", new { id = Model.ID }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

通过引入视图模板文件顶部的 `@model` 语句，您可以指定该视图期望的对象类型。当您创建电影控制器时，Visual Studio 会将 `@model` 声明自动包含到*Details.cshtml*文件的顶部：

```
@model MvcMovie.Models.Movie
```

此 `@model` 声明使得控制器可以将强类型的 `Model` 对象传递给View视图， 从而您可以在视图里访问传递过来的强类型电影Model。 例如，在*Details.cshtml*模板中，每部电影的字段，通过代码传递了 `DisplayNameFor` 和DisplayFor.aspx) HTML Helper通过强类型的 `Model` 对象。Create和Edit方法还有视图模板都在传递电影的强类型模型对象。

看一下*Index.cshtml*视图模版和*MoviesController.cs*中的 `Index` 方法。请注意这些代码是如何在I `ndex` 操作方法中，创建 `List` 对象，并调用 `View` 方法的。

此代码在控制器中传递 `Movies` 列表给视图：

```
public ActionResult Index()
{
    return View(db.Movies.ToList());
}
```

当您创建电影控制器时，Visual Studio会自动包含 `@model` 语句到*Index.cshtml*文件的顶部

```
@model IEnumerable<MvcMovie.Models.Movie>
```

此 `@model` 声明使得控制器可以将强类型的电影列表 `Model` 对象传递给View视图。例如，在*Index.cshtml*模板中，在强类型的 `Model` 对象上使用 `foreach` 语句循环遍历电影列表：

```
@foreach (var item in Model) { <tr> <td>

            @Html.DisplayFor(modelItem => item.Title)

</td> <td>

            @Html.DisplayFor(modelItem => item.ReleaseDate)

</td> <td>

            @Html.DisplayFor(modelItem => item.Genre)

</td> <td>

            @Html.DisplayFor(modelItem => item.Price)

</td> <th>

            @Html.DisplayFor(modelItem => item.Rating)

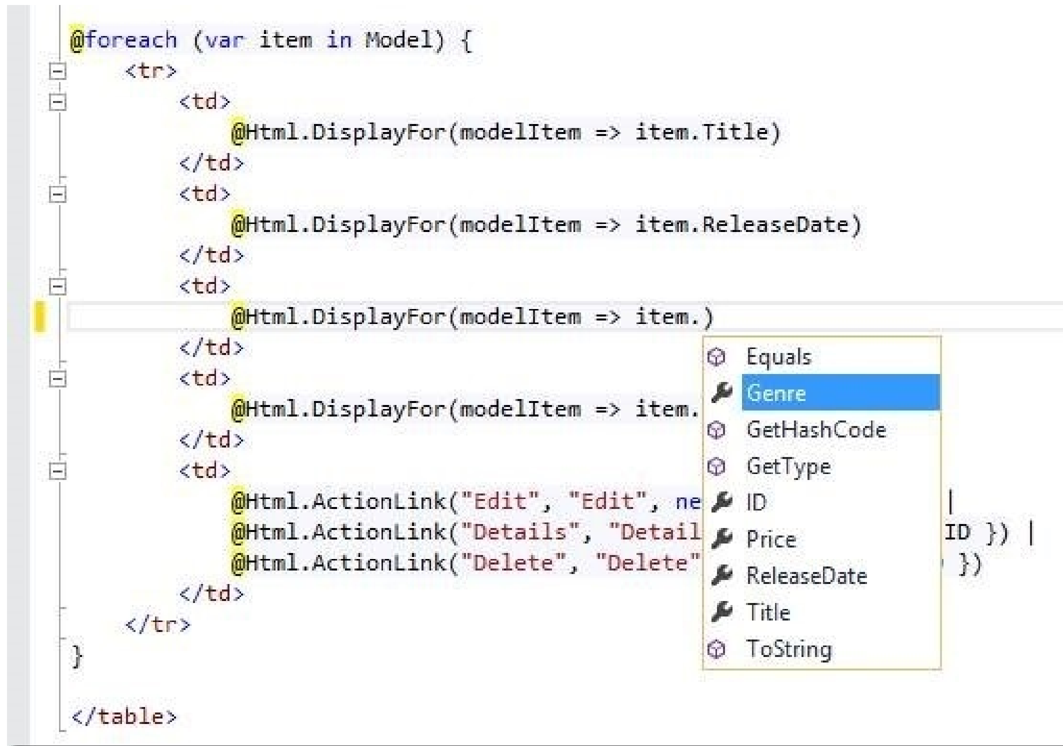
</th> <td>

            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", { id=item.ID })  |
            @Html.ActionLink("Delete", "Delete", { id=item.ID })

</td> </tr>

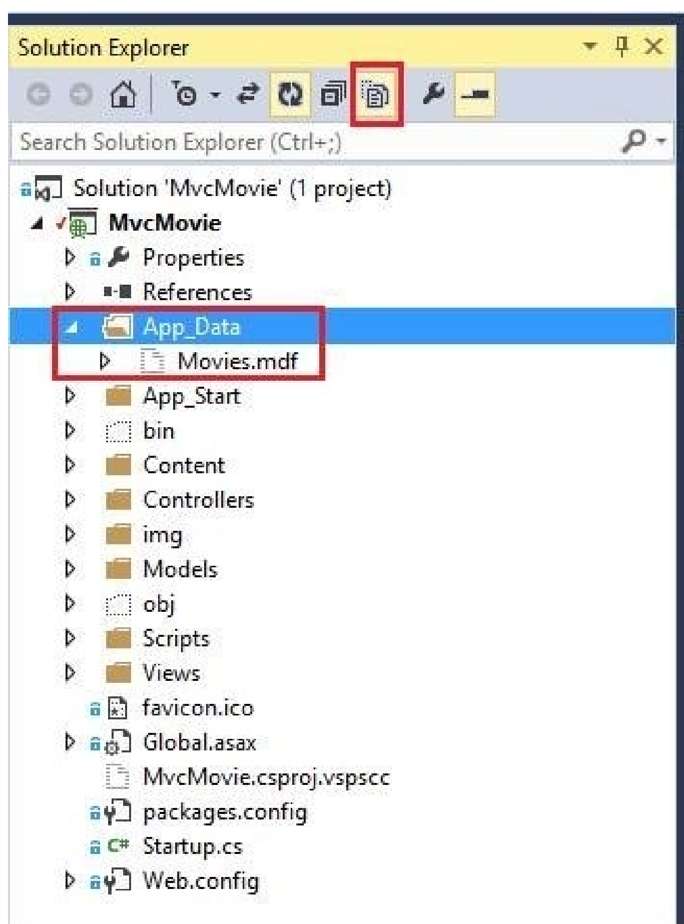
}
```

因为 `Model` 对象是强类型的 （是 `IEnumerable<Movie>` 对象），所以在循环中的每个 `item` 对象的类型是 `Movie` 类型。好处之一是，这意味着您可以在代码编译时进行检查，同时在代码编辑器中支持更加全面的智能感知：



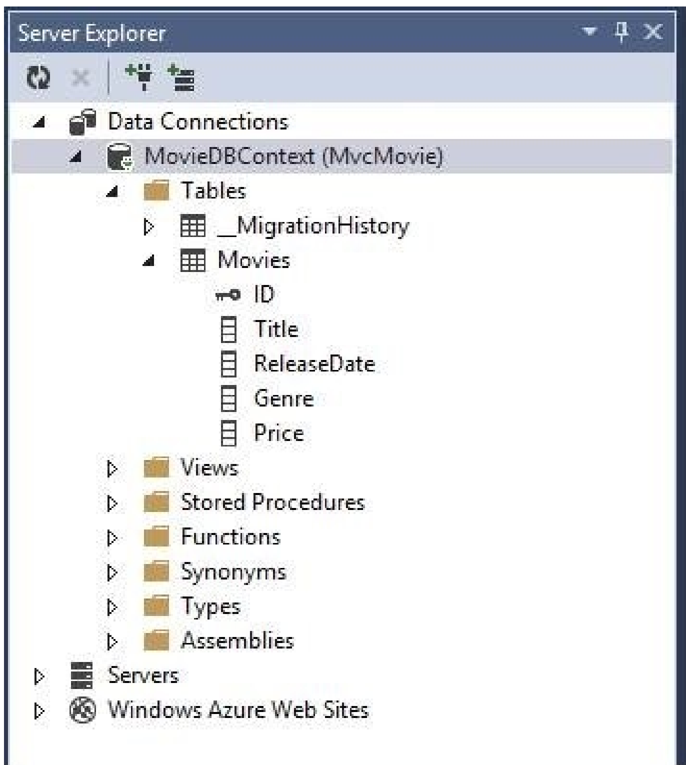
使用SQL Server LocalDB

Entity Framework Code First（代码优先），如果检测到不存在一个数据库连接字符串指向了 `Movies` 数据库，会自动的创建数据库。在*App_Data*文件夹中找一下，您可以验证它已经被创建了。如果您看不到*Movies.mdf*文件，请在解决方案资源管理器工具栏上，单击显示所有文件按钮，单击刷新按钮，然后展开App_Data文件夹。

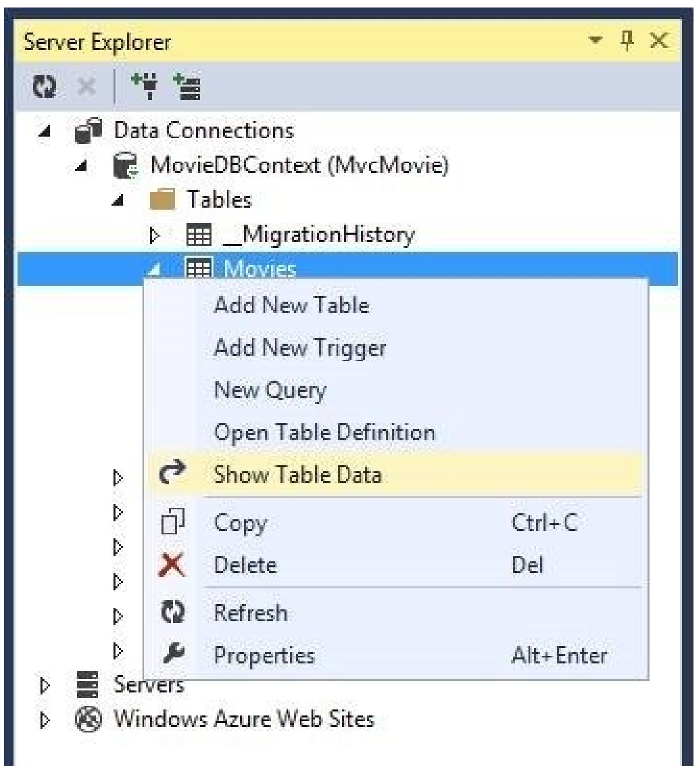


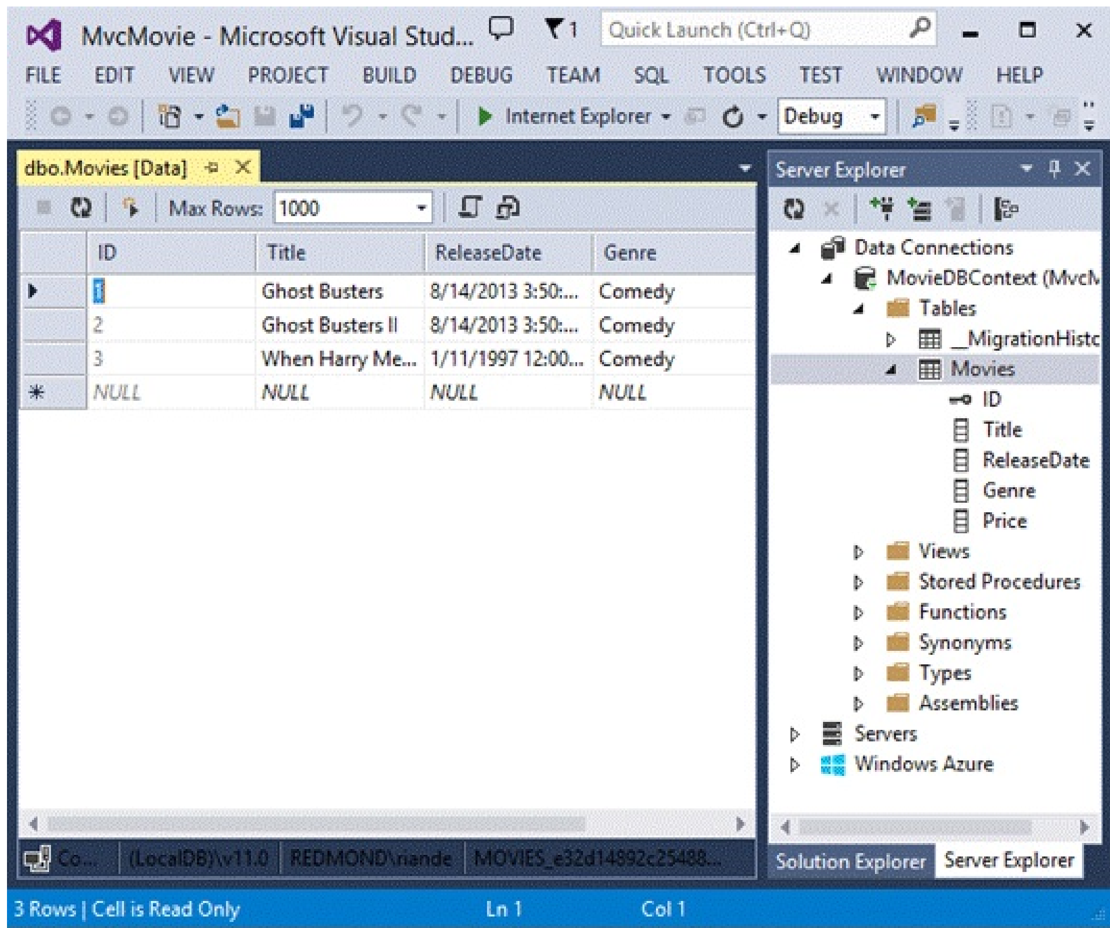
双击*Movies.mdf*打开数据库资源管理器**(SERVER EXPLORER)**，然后展开表文件夹(Tables)以查看电影表。

注意：ID旁边的钥匙图标。默认情况下，EF将创建一个名为ID的主键。欲了解更多EF和MVC信息，请参阅Tom Dykstra's的优秀教程MVC and EF。

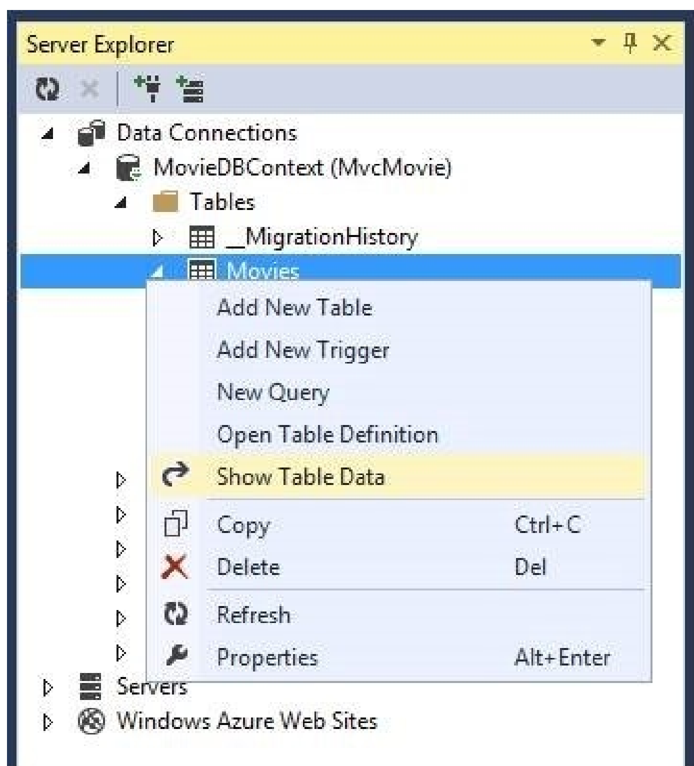


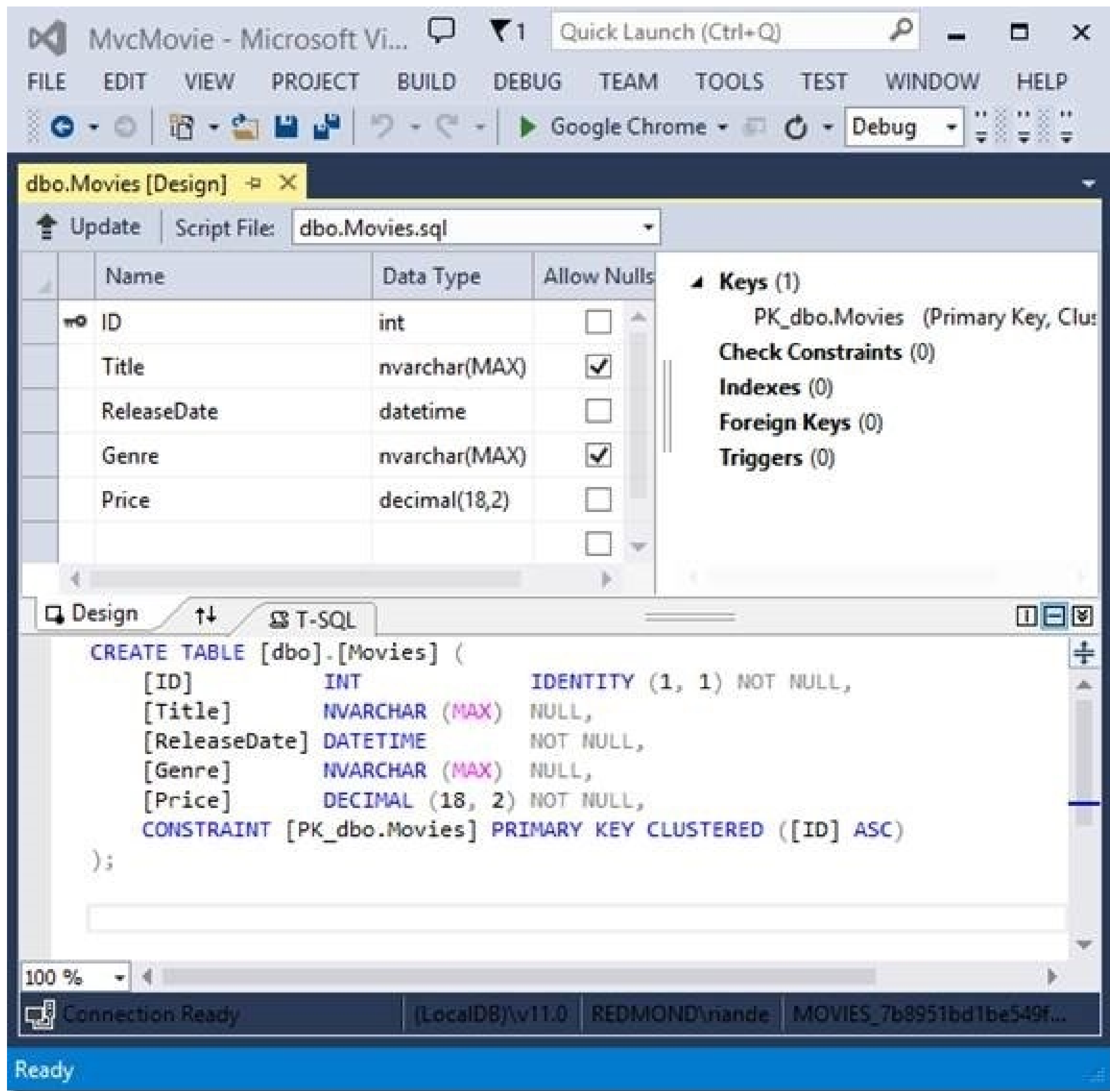
在Movies表上单击鼠标右键，并请选择显示表数据（**Show Table Data**）看您所创建的数据。



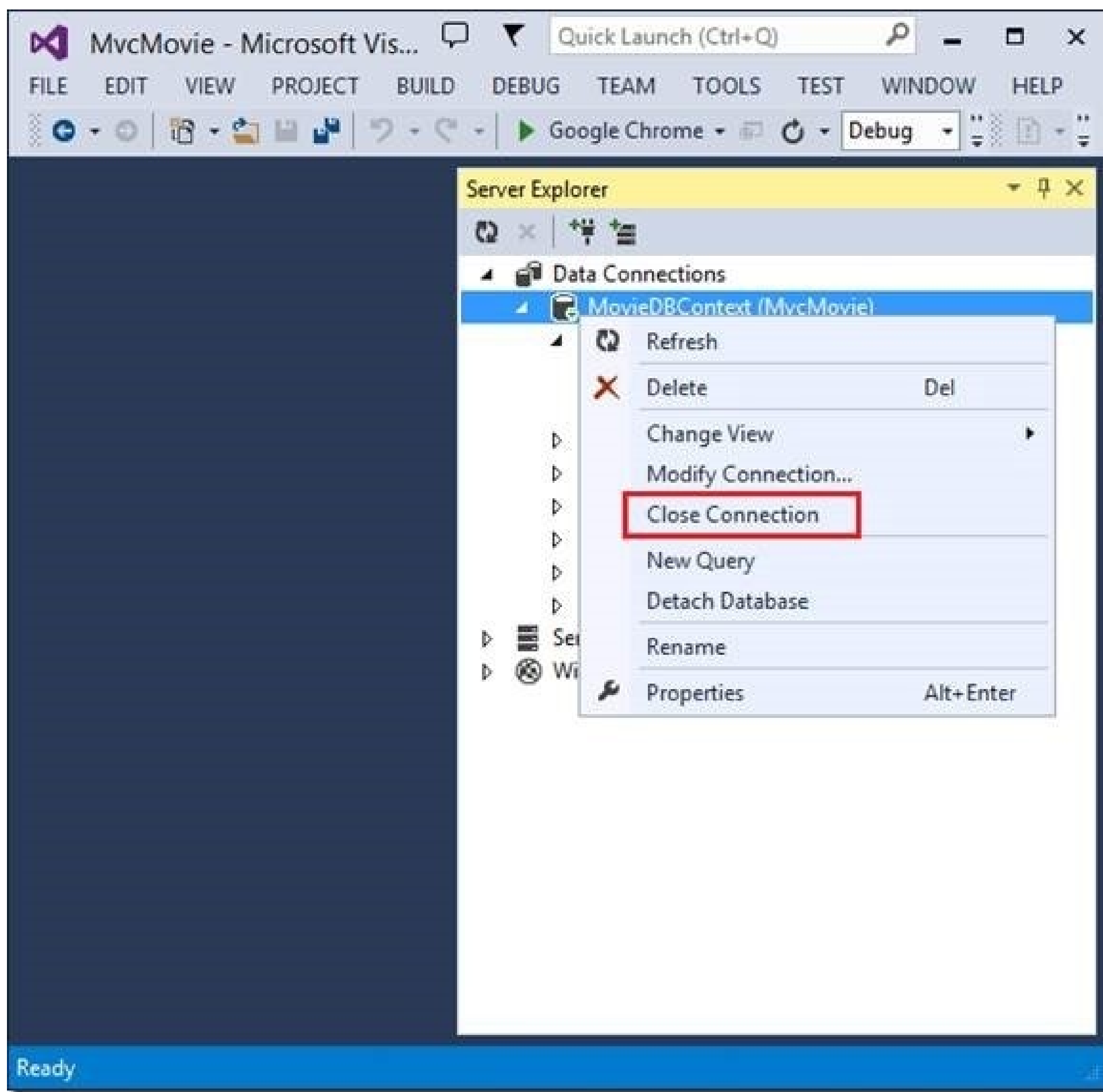


在Movies表上单击鼠标右键，并请选择打开表定义（**Open Table Definition**），您将看到Entity Framework Code First创建的表结构。





注意事项： Movies表映射到Movie类的架构（schema）如何你前面创建的。Entity Framework Code First首先自动为您创造了这个架构（schema）基于Movie class。当您完成后，通过右击MovieDBContext，并选择关闭连接。（如果你不关闭连接，下一次运行项目，你可能会得到一个错误）。



现在，您可以在这个简单列表页面里：显示、编辑、更新、删除数据库里的数据了。在下一次的教程中，我们会继续看看scaffolded自动生成的其它代码。并添加一个 `SearchIndex` 方法和 `SearchIndex` 视图，使您可以在数据库中搜索电影了。更多关于Entity Framework with MVC, see Creating an Entity Framework Data Model for an ASP.NET MVC Application.

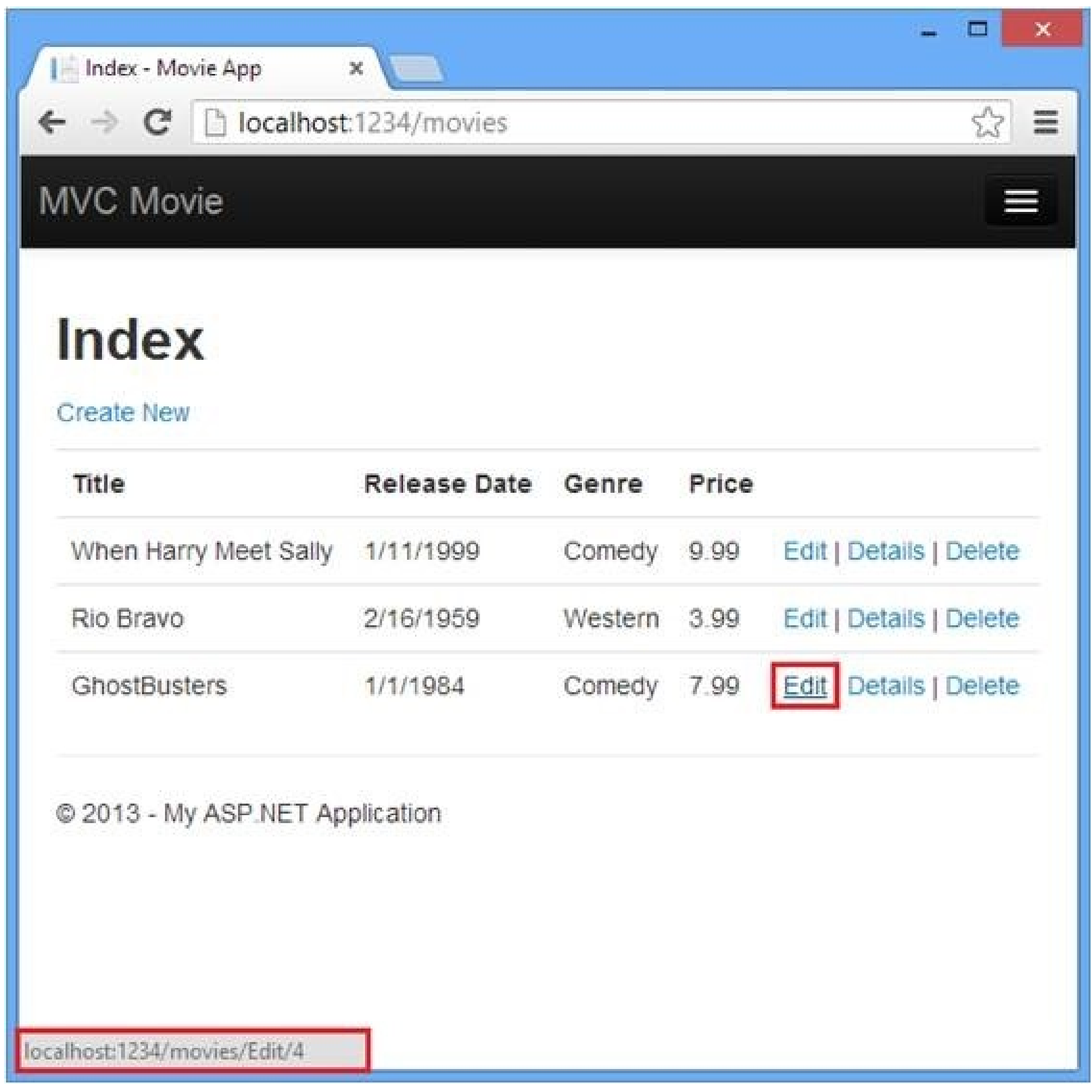
ASP.NET MVC 5 - 验证编辑方法(Edit method)和编辑视图(Edit view)

在本节中，您将验证电影控制器生成的编辑方法(Edit action methods)和视图。但是首先将修改点代码，使得发布日期属性（ReleaseDate）看上去更好。打开*Models \ Movie.__cs*文件，并添加高亮行如下所示：

```
using System; using System.ComponentModel.DataAnnotations; using System.Data.Entity; name
{ public class Movie
{ public int ID { get; set; } public string Title { get; set; }
[Display(Name = "Release Date")]
[DataType(DataType.Date)]
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)] public
} public class MovieDBContext : DbContext
{ public DbSet<Movie> Movies { get; set; }
}
}
```

在接下来的教程中，我们将讨论DataAnnotations。Display属性指明要显示的字段的名称（在本例中“Release Date”来代替“ReleaseDate”）。DataType属性用于指定类型的数据，在本例它是一个日期，所以不会显示存放在该字段时间详情。DisplayFormat属性在Chrome浏览器里有一个bug：呈现的日期格式不正确。

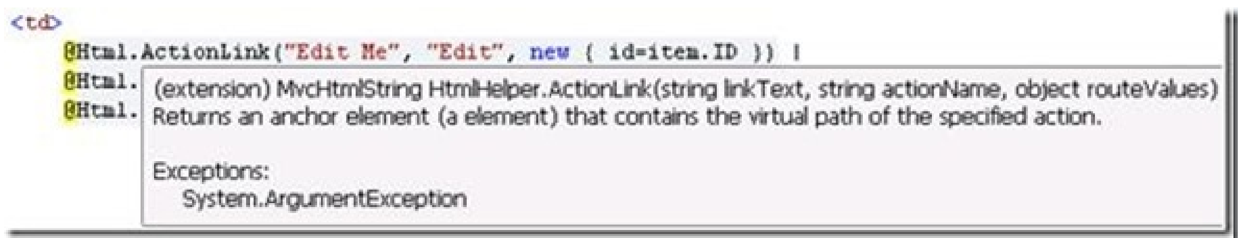
在浏览器地址栏里追加/Movies， 浏览到Movies页面。并进入编辑**(Edit)**页面。



Edit（编辑）**链接是由*Views\Movies\Index.cshtml*视图

中的Html.ActionLink.aspx)方法所生成的

@Html.ActionLink("Edit", "Edit", new { id=item.ID })



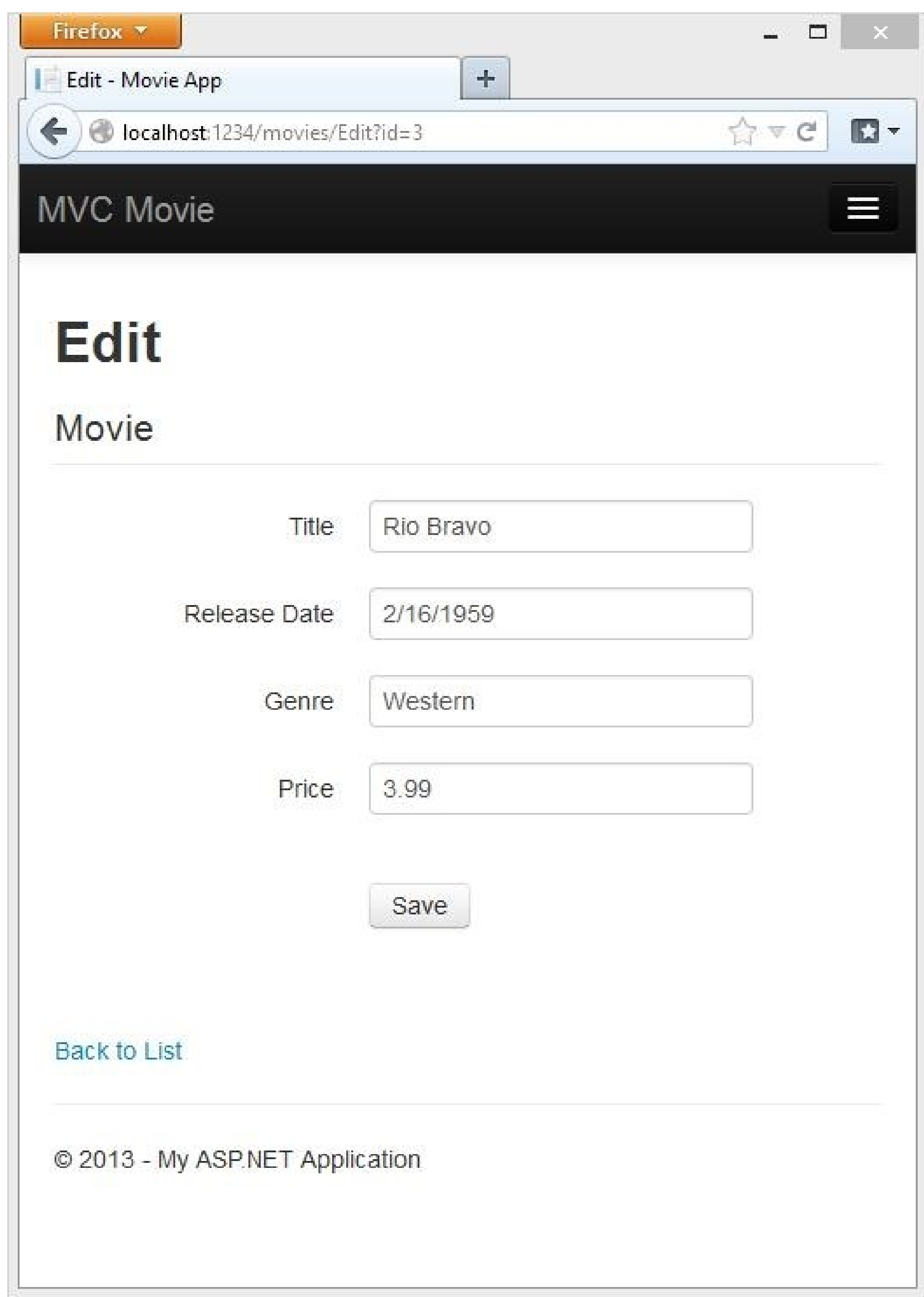
`Html` 对象是一个Helper, 以属性的形式在System.Web.Mvc.WebViewPage.aspx)基类上公开。 ActionLink是一个帮助方法(Helper)，便于动态生成指向Controller中操作方法 的HTML超链接链接。 `ActionLink` 方法的第一个参数是想要呈现的链接文本 （例如， `<a>Edit Me</a>`)。第二个参数是要调用的操作方法的名称（在本例中， Edit方法）。最后一个参数是一个匿名对象（anonymous object），用来生成路由数据 （在本例中，ID 为 4 的）。

在上图中所生成的链接是http://localhost:xxxxx/Movies/Edit/4。默认的路由（在*App_Start\RouteConfig.cs* 中设定） 使用的 URL 匹配模式为: `{controller}/{action}/{id}` 。因此，ASP.NET 将*http://localhost:xxxxx/Movies/Edit/4*转化到 `Movies` 控制器中 `Edit` 操作方法，参数 `ID` 等于 4 的请求。查看*App_Start\RouteConfig.cs*文件中的以下代码。

MapRoute方法是使用HTTP请求路由查找到正确的控制器（controller）和行动方法，并提供了可选ID的参数。MapRoute方法也被用于通过HtmlHelpers.aspx)如ActionLink的控制器，操作方法及任何路由数据，以生成URL。

```
public static void RegisterRoutes(RouteCollection routes)
{
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
routes.MapRoute(
name: "Default",
url: "{controller}/{action}/{id}",
defaults: new { controller = "Home", action = "Index",
id = UrlParameter.Optional }
);
}
```

您还可以使用QueryString来传递操作方法的参数。例如，URL: *http://localhost:xxxxx/Movies/Edit?ID=__3*还会将参数 `ID` 为 3的请求传递给 `Movies` 控制器的 `Edit` 操作方法。



打开 `Movies` 控制器。如下所示的两个 `Edit` 操作方法。

```
// GET: /Movies/Edit/5

public ActionResult Edit(int? id)

{ if (id == null)

{ return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Movie movie = db.Movies.Find(id); if (movie == null)

{ return HttpNotFound();

} return View(movie);

} // POST: /Movies/Edit/5 // To protect from overposting attacks, please enable the speci
 [HttpPost]

[ValidateAntiForgeryToken] public ActionResult Edit([Bind(Include="ID,Title,ReleaseDate,G

{ if (ModelState.IsValid)

{

db.Entry(movie).State = EntityState.Modified;

db.SaveChanges(); return RedirectToAction("Index");

} return View(movie);

}
```

注意，第二个 `Edit` 操作方法的上面有HttpPost属性。此属性指定了 `Edit` 方法的重载，此方法仅被POST 请求所调用。您可以将HttpGet属性应用于第一个编辑方法，但这是不必要的，因为它是默认的属性。（操作方法会被隐式的指定为 `HttpGet` 属性，从而作为 `HttpGet` 方法。） 绑定（Bind.aspx)）属性是另一个重要安全机制，可以防止黑客攻击(从over-posting数据到你的模型)。您应该只包含在bind属性属性，您想要更改。您可以阅读有关在我overposting security note。我们将在本教程中使用的简单模型，模型中绑定所有数据。ValidateAntiForgeryToken.aspx)属性是用来防止伪造的请求，并配对@Html.AntiForgeryToken.aspx)()文件 (*Views\Movies\Edit.cshtml*)，如下图所示，部分在编辑view文件：

```
@model MvcMovie.Models.Movie
@{
ViewBag.Title = "Edit";
} <h2>Edit</h2> @using (Html.BeginForm())
{
@Html.AntiForgeryToken() <div class="form-horizontal">
<h4>Movie</h4>
<hr /> @Html.ValidationSummary(true)
@Html.HiddenFor(model => model.ID) <div class="form-group"> @Html.LabelFor(model => model
@Html.ValidationMessageFor(model => model.Title) </div>
</div>
```

@Html.AntiForgeryToken() 生成隐藏的窗体, 防伪令牌必须匹配的的Movies控制器的Edit方法。在我的教程XSRF/CSRF Prevention in MVC，你可以读到更多关于跨站点请求伪造（也称为XSRF或CSRF）。

`HttpGet` `Edit` 方法会获取电影ID参数、 查找影片使用Entity Framework 的Find.aspx)方法，并返回到选定影片的编辑视图。如果不带参数调用 `Edit` 方法，ID 参数被指定为默认值零。如果找不到一部电影，则返回HttpNotFound.aspx) 。当scaffolding自动创建编辑视图时，它会查看 `Movie` 类并为类的每个属性创建用于Render的 `<label>` 和 `<input>` 的元素。下面的示例为visual studio scaffolding自动创建的编辑视图：

```
@model MvcMovie.Models.Movie
@{
ViewBag.Title = "Edit";
} <h2>Edit</h2> @using (Html.BeginForm())
{
@Html.AntiForgeryToken() <div class="form-horizontal">
<h4>Movie</h4>
<hr /> @Html.ValidationSummary(true)
@Html.HiddenFor(model => model.ID) <div class="form-group"> @Html.LabelFor(model => model
@Html.ValidationMessageFor(model => model.Title) </div>
</div>
<div class="form-group"> @Html.LabelFor(model => model.ReleaseDate, new { @class = "contr
@Html.ValidationMessageFor(model => model.ReleaseDate) </div>
</div> @*Genre and Price removed for brevity.*@ <div class="form-group">
<div class="col-md-offset-2 col-md-10">
<input type="submit" value="Save" class="btn btn-default" />
</div>
</div>
</div> } <div> @Html.ActionLink("Back to List", "Index") </div> @section Scripts {
@Scripts.Render("~/bundles/jqueryval")
}
```

注意，视图模板在文件的顶部有 `@model MvcMovie.Models.Movie` 的声明，这将指定视图期望的模型类型为 `Movie` 。

scaffolded自动生成的代码，使用了*Helper*方法的几种简化的 HTML 标记。`Html.LabelFor` .aspx) 用来显示字段的名称（"Title"、"ReleaseDate"、"Genre"或"Price")。`Html.EditorFor` .aspx) 用来呈现 HTML `<input>` 元素。`Html.ValidationMessageFor` .aspx) 用来显示与该属性相关联的任何验证消息。

运行该应用程序，然后浏览URL，*/Movies*。单击**Edit**链接。在浏览器中查看页面源代码。HTML Form中的元素如下所示：

```
<form action="/movies/Edit/4" method="post">
<input name="__RequestVerificationToken" type="hidden" value="UxY6bkQyJCXO3Kn5AXg-6TXxOj6
<legend>Movie</legend>
<input data-val="true" data-val-number="The field ID must be a number." data-val-required
<div class="control-group">
```

```
<label class="control-label" for="Title">Title</label>
<div class="controls">
<input class="text-box single-line" id="Title" name="Title" type="text" value="GhostBuste
<span class="field-validation-valid help-inline" data-valmsg-for="Title" data-valmsg-repl
</div>
</div>
<div class="control-group">
<label class="control-label" for="ReleaseDate">Release Date</label>
<div class="controls">
<input class="text-box single-line" data-val="true" data-val-date="The field Release Date
<span class="field-validation-valid help-inline" data-valmsg-for="ReleaseDate" data-valms
</div>
</div>
<div class="control-group">
<label class="control-label" for="Genre">Genre</label>
<div class="controls">
<input class="text-box single-line" id="Genre" name="Genre" type="text" value="Comedy" />
<span class="field-validation-valid help-inline" data-valmsg-for="Genre" data-valmsg-repl
</div>
</div>
<div class="control-group">
<label class="control-label" for="Price">Price</label>
<div class="controls">
<input class="text-box single-line" data-val="true" data-val-number="The field Price must
<span class="field-validation-valid help-inline" data-valmsg-for="Price" data-valmsg-repl
</div>
</div>
<div class="form-actions no-color">
<input type="submit" value="Save" class="btn" />
</div>
</fieldset>
</form>
```

被 `<form>` HTML 元素所包括的 `<input>` 元素会被发送到，<form>的 `action` 属性所设置的URL:*/Movies/Edit*。单击**Save**按钮时，from数据将会被发送到服务器。第二行显示隐藏XSRF通过@Html.AntiForgeryToken()调用生成的令牌。

处理 POST 请求

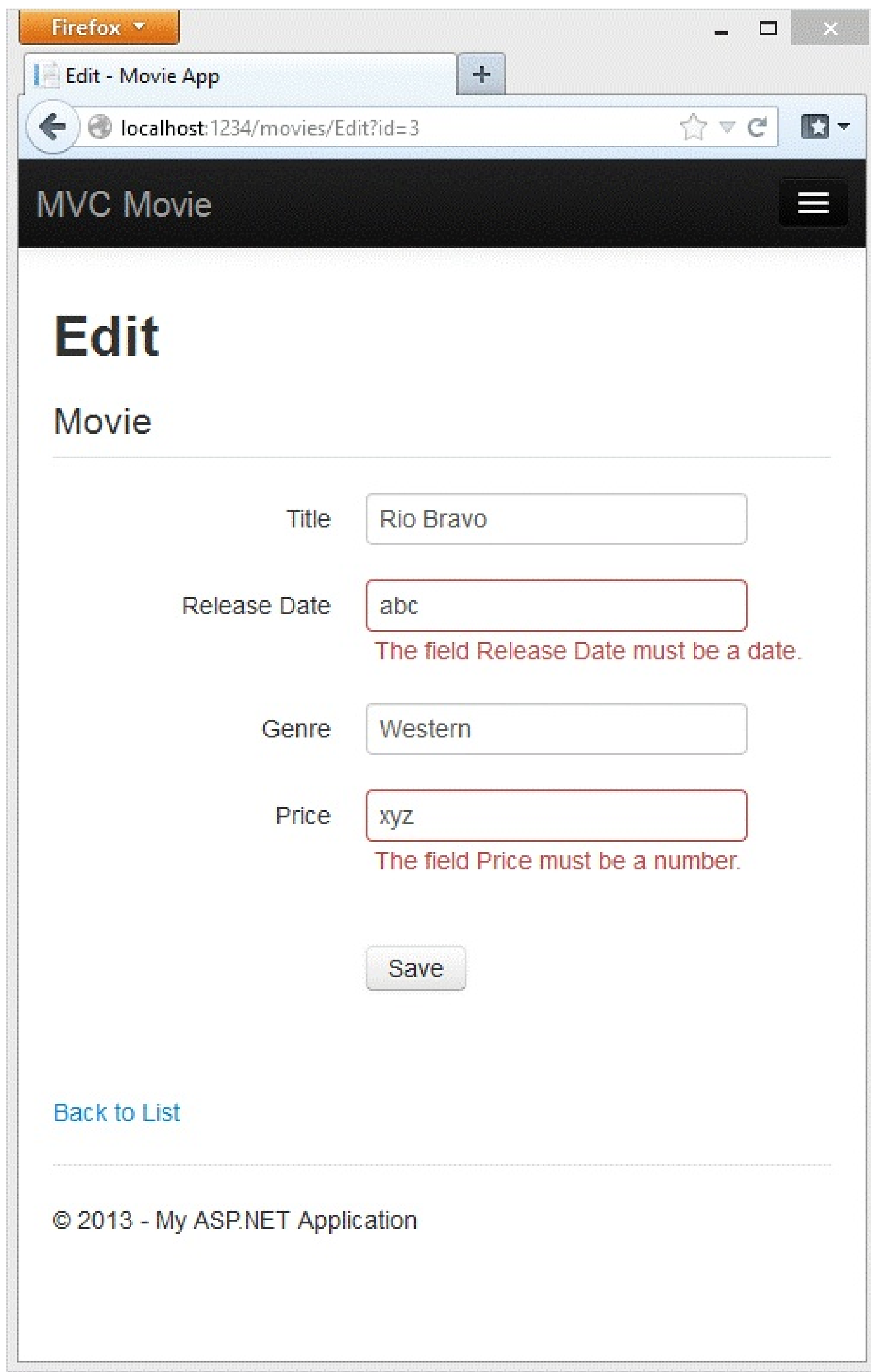
下面的代码显示了 `Edit` 操作方法的 `HttpPost` 处理：

```
[HttpPost]
[ValidateAntiForgeryToken] public ActionResult Edit([Bind(Include="ID,Title,ReleaseDate,G
{ if (ModelState.IsValid)
{
db.Entry(movie).State = EntityState.Modified;
db.SaveChanges(); return RedirectToAction("Index");
} return View(movie);
}
```

ASP.NET MVC model binder

接收form所post的数据，并转换所接收的Movie请求数据从而创建一个 `Movie` 对象。 `ModelState.IsValid` 方法用于验证提交的表单数据是否可用于修改（编辑或更新）一个 `Movie` 对象。如果数据是有效的电影数据，将保存到数据库的 `Movies` 集合 `(MovieDBContext` 实例)。通过调用 `MovieDBContext` 的 `SaveChanges` 方法，新的电影数据会被保存到数据库。数据保存之后，代码会把用户重定向到 `MoviesController` 类的 `Index` 操作方法， 页面将显示电影列表，同时包括刚刚所做的更新。

一旦客户端验证确定某个字段的值是无效的，将显示出现错误消息。如果禁用JavaScript，则不会有客户端验证，但服务器将检测回传的值是无效的，而且将重新显示表单中的值与错误消息。在本教程的后面，我们验证更详细的审查。*Edit.cshtml*视图模板中的 `Html.ValidationMessageFor` Helper将用来显示相应的错误消息。

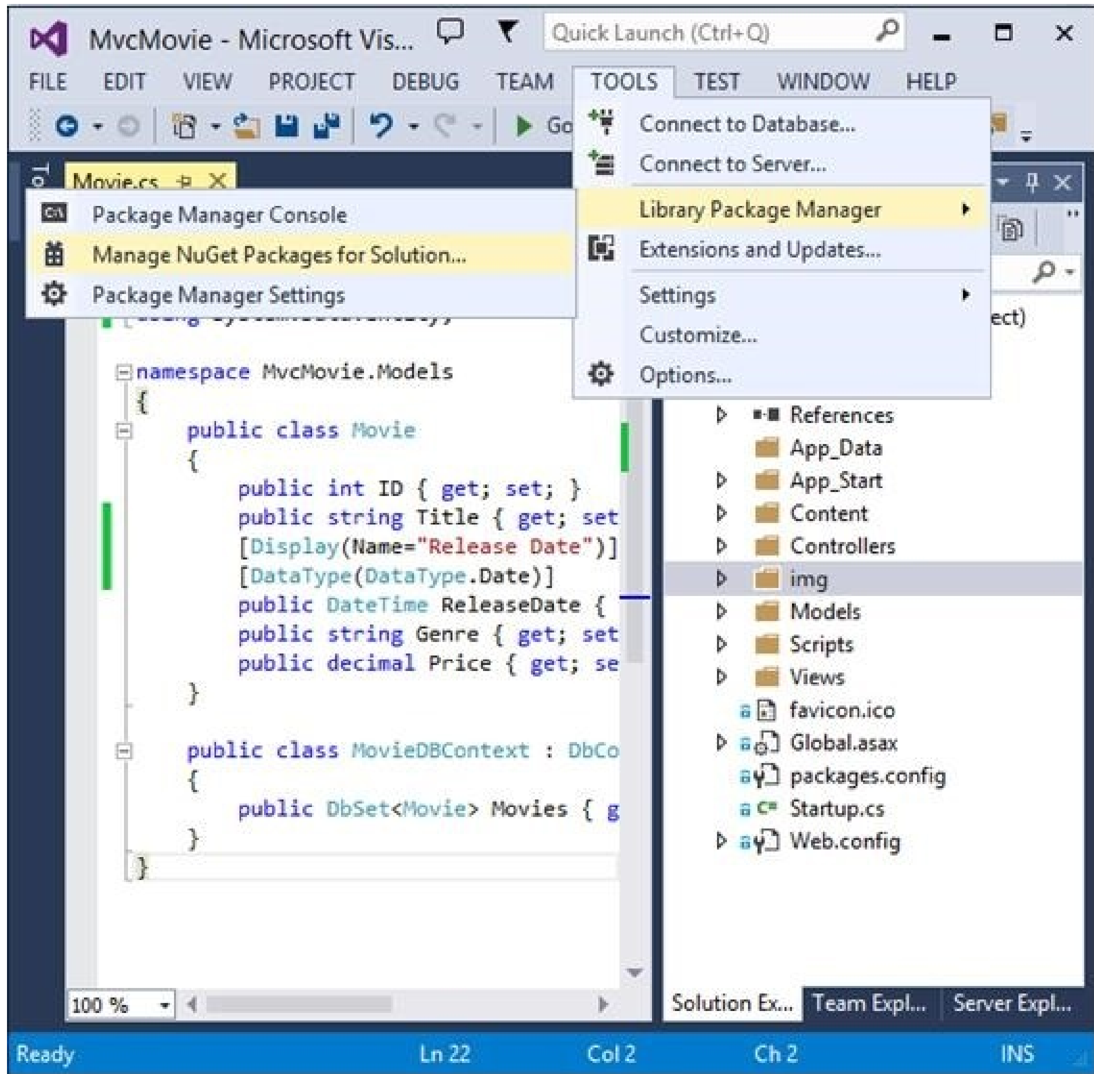


所有HttpGet方法遵循类似的模式。他们得到一个电影对象（或对象列表中，如本案例的Index），并把模型数据传递给视图。Create方法传递一个空的影片对象给Create视图。所有的create, edit, delete方法，或其他的方法： 用HttpPost重载的方法修改数据。修改数据在HTTP GET方法, 存在安全风险，如博客文章ASP.NET MVC Tip #46 – Don't use Delete Links because they create Security Holes. 在HTTP GET方法中修改数据也违反HTTP的最佳实践和REST模式架构，指明GET请求不应该改变你的应用程序的状态。换句话说，执行GET操作应该是一个安全，操作，无任何副作用，不会修改你的持久化数据。

如果您的电脑是是US-English的语言设置，可以跳过这一节，直接进入下一个教程。

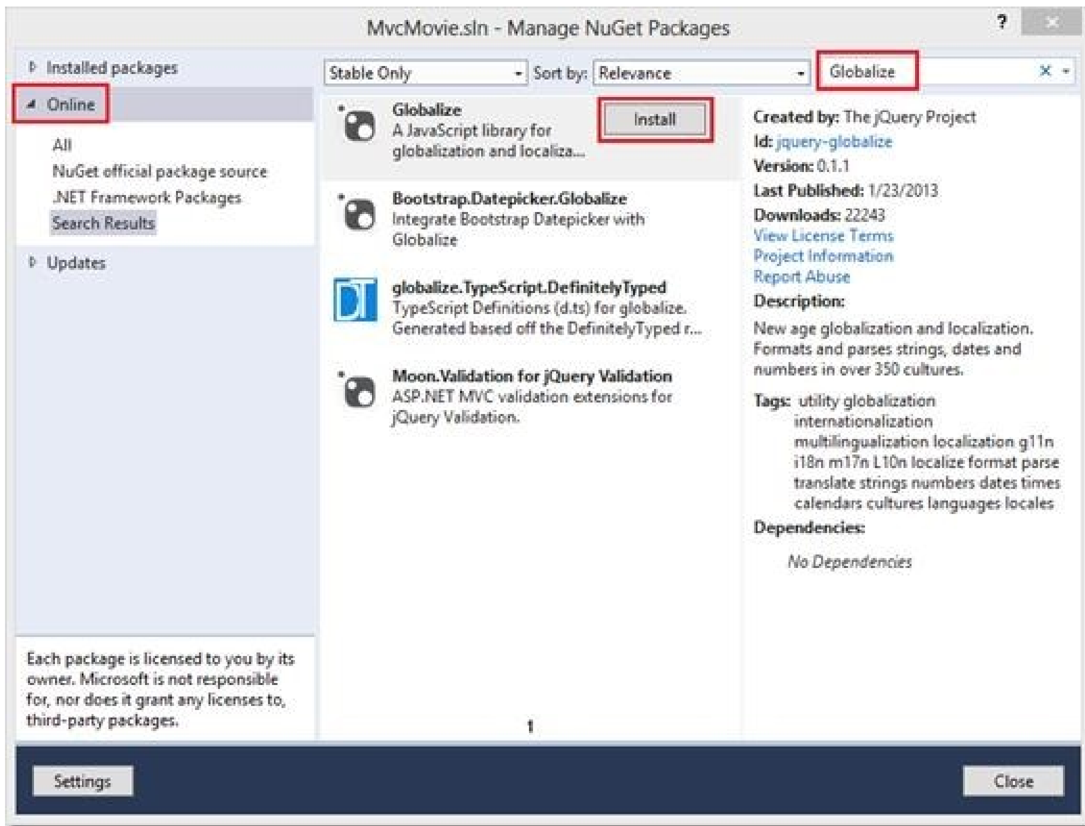
注意，为了使jQuery支持使用逗号的非英语区域的验证 ，需要设置逗号（"，"）来表示小数点，你需要引入*globalize.js*并且你还需要具体的指定*cultures/globalize.cultures.js*文件 （地址在https://github.com/jquery/globalize) 在 JavaScript 中可以使用 `Globalize.parseFloat` 。你可以从NuGet中安装非英语的jQuery的验证、插件。 （如果您使用的是英语语言环境，不要安装全球化 (Globalize)。）

1. 在工具（**Tools**）菜单，点击库程序包管理器（ **Library Package Manager**）， 选择解决方案程序包管理器（Manage NuGet Packages for Solution**）.



2. 在左边面板上，选择联机库（**Online**，**见下图）

3. 在搜索已安装库（ **Search Installed packages** ）， 输入 *Globalize*搜索



点击安装(**Install**). JavaScript脚本 *\jquery.globalize\globalize.js* 文件将会添加到您的当前工程下. 脚本*\jquery.globalize\cultures* 文件夹的下面会包含很多不同文化的JavaScript文件

注意事项：安装这个包，预计花费5分钟时间（取决于您的网速）.

下面的代码展示了在"FR-FR" Culture下的 Views\Movies\Edit.cshtml 视图：

```
@section Scripts {
@Scripts.Render("~/bundles/jqueryval") <script src="~/Scripts/jquery.globalize/globalize.
<script src="~/Scripts/jquery.globalize/cultures/globalize.culture.fr-FR.js"></script>
<script> $.validator.methods.number = function (value, element) { return this.optional(el
!isNaN(Globalize.parseFloat(value));
}
$(document).ready(function () {
Globalize.culture('fr-FR');
}); </script>
<script> jQuery.extend(jQuery.validator.methods, {
range: function (value, element, param) { //Use the Globalization plugin to parse the val
var val = $.global.parseFloat(value); return this.optional(element) || (
val >= param[0] && val <= param[1]);
}
}); </script>
<script> $.validator.methods.date = function (value, element) { return this.optional(elem
} </script> }
```

为了避免在每一个编辑视图重复这段代码，你可以将它移动到布局文件。要优化脚本下载，看我的教程Bundling and Minification。欲了解更多信息，请，ASP.NET MVC 3 Internationalization和ASP.NET MVC 3 Internationalization - Part 2 (NerdDinner)。

作为一个临时解决办法，如果您不能验证当前的区域设置，可以强制你的计算机使用US English，或者你可以在浏览器中禁用JavaScript。为了强制您的电脑使用美国英语，你可以在项目根目录Web.config文件里面添加的全球化设置。

下面的代码演示设置为美国英语的全球化文化设置。

```
<system.web>
<globalization culture ="en-US" />
<!--elements removed for clarity-->
</system.web>
```

在接下来的教程，我们将实现搜索功能。

添加一个搜索方法(Search Method)和搜索视图(Search View)

在本节中，您将添加Index操作方法，可以让你按照电影流派(genre)或名称搜索电影。

升级 Index窗体

我们开始在方法现有MoviesController类中，更新Index方法。代码如下：

```
public ActionResult Index(string searchString)
{ var movies = from m in db.Movies select m; if (!String.IsNullOrEmpty(searchString))
{
        movies = movies.Where(s => s.Title.Contains(searchString));
} return View(movies);
}
```

Index方法的第一行创建以下的LINQ查询，以选择看电影：

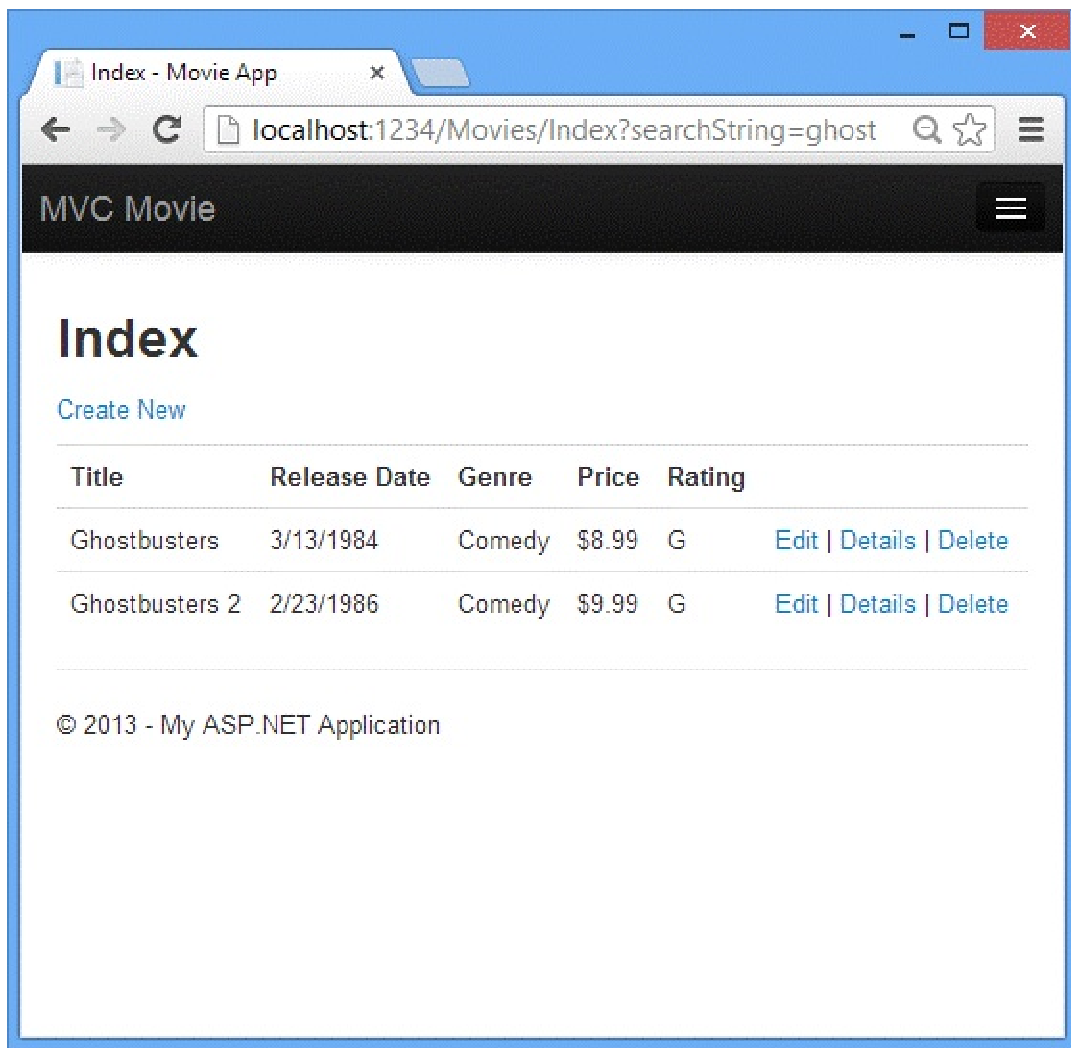
```
var movies = from m in db.Movies select m;
如果searchString参数包含一个字符串，可以使用下面的代码，修改电影查询要筛选的搜索字符串： if (!String.I
{
        movies = movies.Where(s => s.Title.Contains(searchString));
}
```

上面 `s => s.Title` 代码是一个Lambda 表达式。Lambda 是基于方法的LINQ查询，例如上面的where查询。在上面的代码中使用了标准查询参数运算符的方法。当定义LINQ查询或修改查询条件时，如调用 `Where` 或 `OrderBy` 方法时，不会执行 LINQ 查询。相反，查询执行会被延迟，这意味着表达式的计算延迟，直到取得实际的值或调用 `ToList` 方法。在Search示例中，*Index.cshtml*视图中执行查询。有关延迟的查询执行的详细信息，请参阅Query Execution.

注：Contains 方法是运行在的数据库，而不是C#代码上面。在数据库中，Contains映射到to SQL LIKE，这是大小写不敏感的。

现在，您可以实现Index视图并将其显示给用户。

运行这个应用程序和导航到 */Movies/Index*。追加一个查询字符串，URL如？searchString=ghost。筛选的影片会被显示。



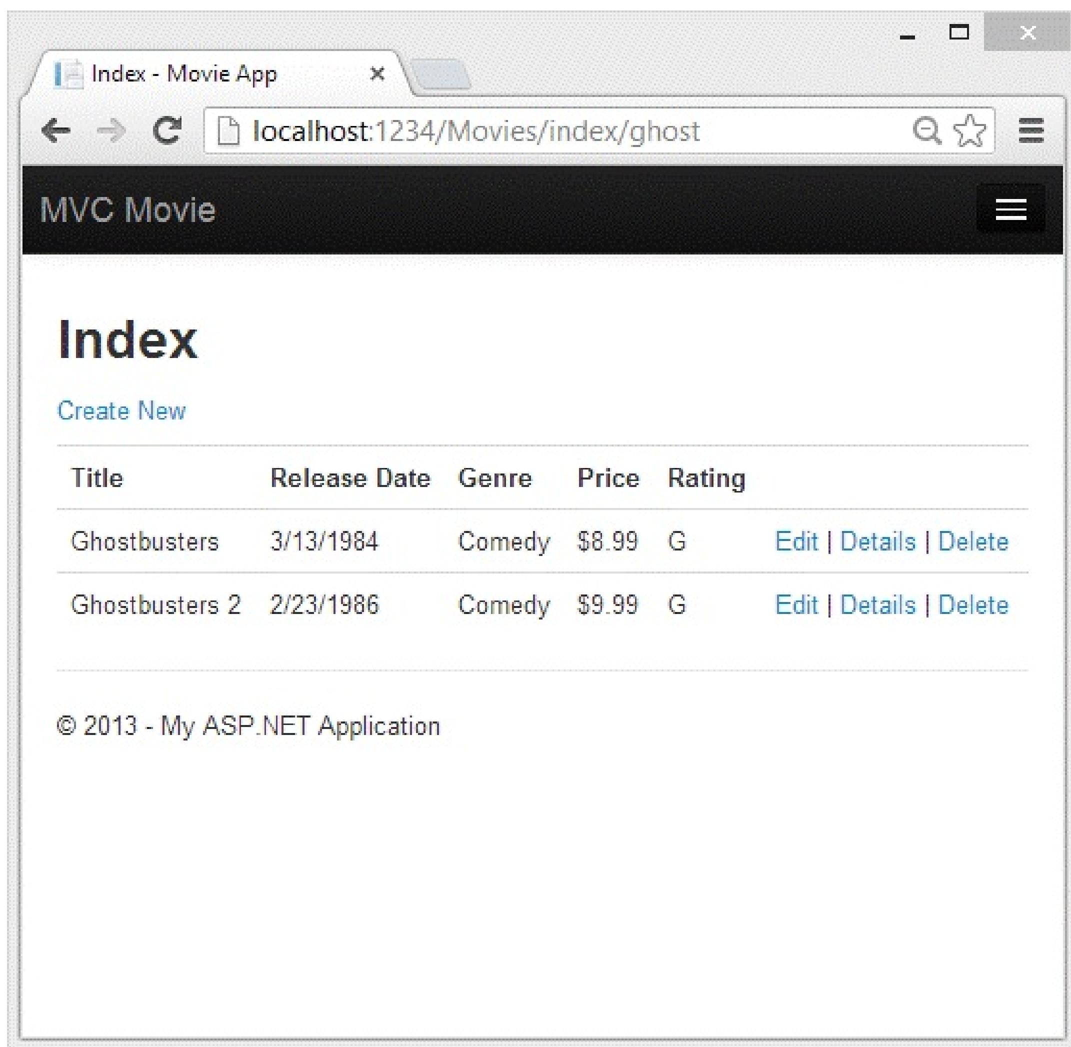
如果你改变了Index方法签名参数名为id的，这个id参数将匹配{ id }的占位符。App_Start\RouteConfig.cs文件中设置的缺省路由定义如下。

{controller}/{action}/{id}

原来的 Index 方法看起来如下所示:

```
public ActionResult Index(string searchString)
{ var movies = from m in db.Movies select m; if (!String.IsNullOrEmpty(searchString))
{
        movies = movies.Where(s => s.Title.Contains(searchString));
} return View(movies);
}
修改后的 Index 方法看起来如下所示: public ActionResult Index(string id)
{ string searchString = id; var movies = from m in db.Movies select m; if (!String.IsNull
{
        movies = movies.Where(s => s.Title.Contains(searchString));
} return View(movies);
}
```

现在，您可以通过路由数据（URL段）的标题搜索了，而不是作为查询字符串值，截图如下：

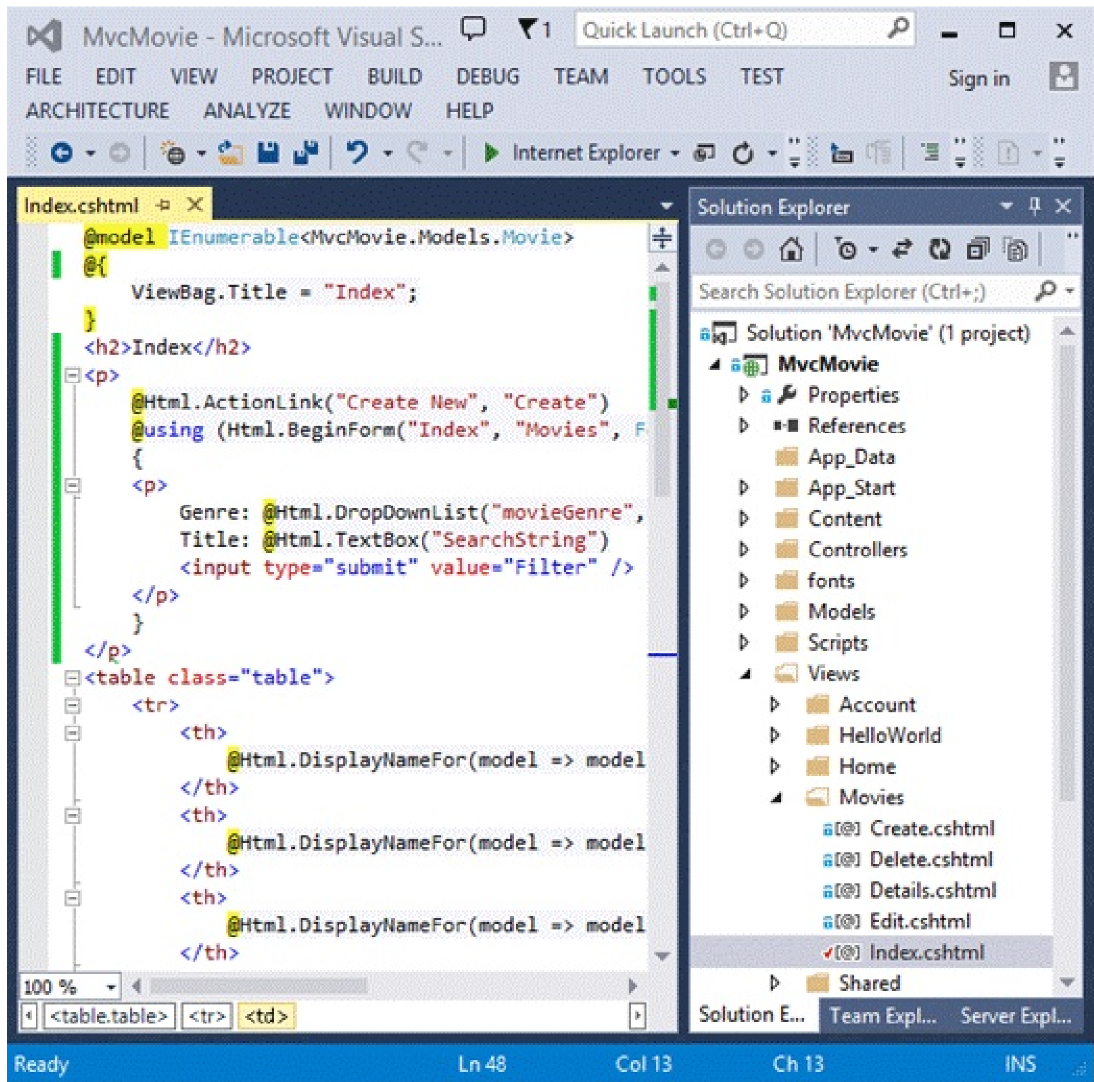


然而，你不能期望用户可以每次要搜索一部电影都会去修改URL。所以，现在你将添加用户界面，帮助他们来过滤影片。如果你改变Index方法来测试如何通过路由绑定ID参数的签名，Index方法需要一个字符串参数searchString：

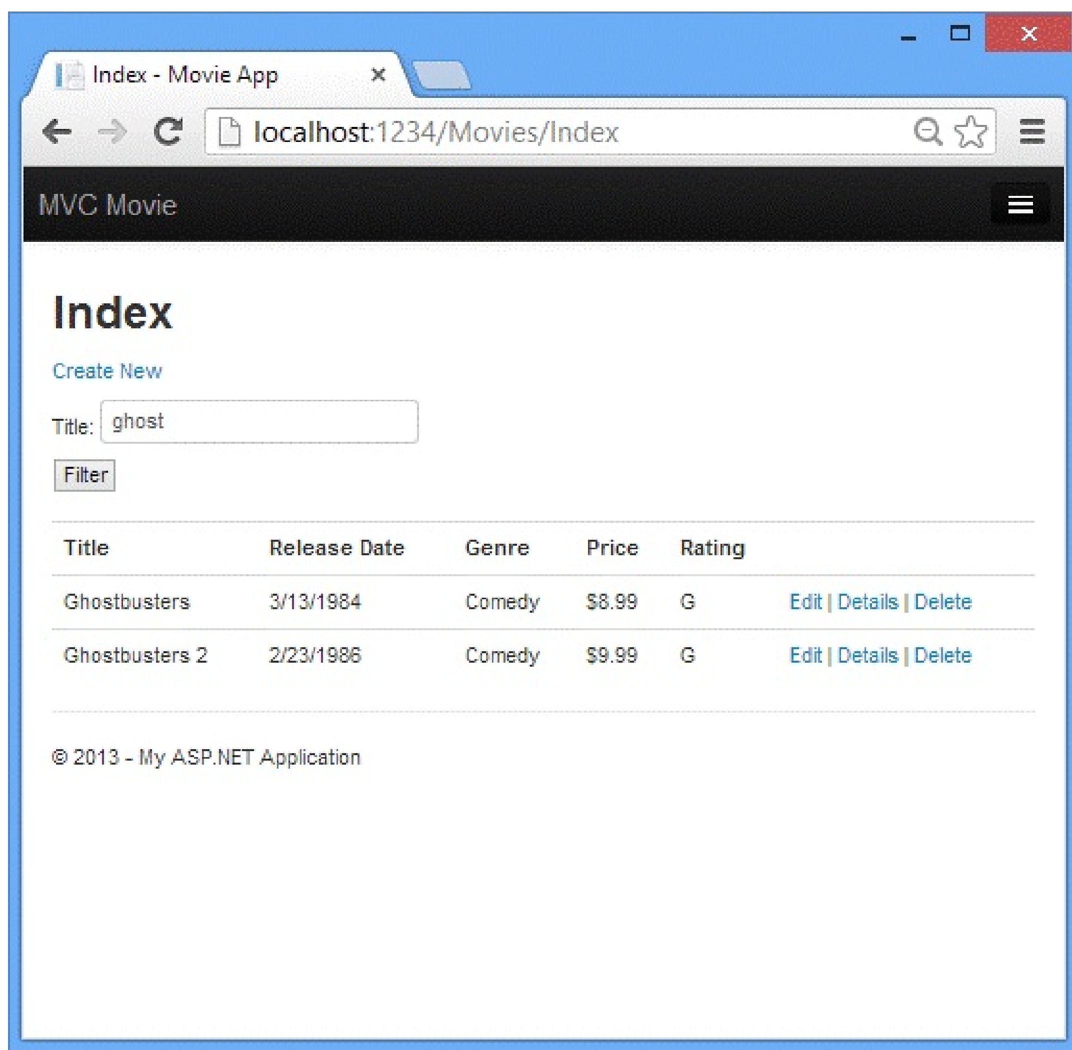
```
public ActionResult Index(string searchString)
{
var movies = from m in db.Movies
select m;
if (!String.IsNullOrEmpty(searchString))
{
        movies = movies.Where(s => s.Title.Contains(searchString));
}
return View(movies);
}
打开文件 Views\Movies\Index.cshtml, 在这段代码@Html.ActionLink("Create New", "Create")后之后,
@model IEnumerable<MvcMovie.Models.Movie> @{
    ViewBag.Title = "Index";
} <h2>Index</h2>
<p> @Html.ActionLink("Create New", "Create")
@using (Html.BeginForm()){ <p> Title: @Html.TextBox("SearchString") <br />
<input type="submit" value="Filter" /></p> } </p>
```

Html.BeginForm辅助会创建一个<form>标签。当用户通过点击“过滤器”按钮，提交表单，Html.BeginForm助手会导致窗体post到它本身。

Visual Studio2013中有一个很好的改善: 显示和编辑视图文件时。当你运行应用程序打开视图文件时，Visual Studio2013的将调用正确的控制器操作方法来展示视图。



在Visual Studio中打开使用Index视图（在上面的图片所示），点击Ctr F5或F5运行应用程序，然后试试搜索一部电影。



该Index 方法的HttpPost没有重载。 你不需要它，因为该方法不改变application的状态，只是过滤数据。

您可以添加以下httppost Index方法。在这种情况下，函数调用将匹配的HttpPost Index方法，的HttpPost Index方法运行的如下面的图片所示。

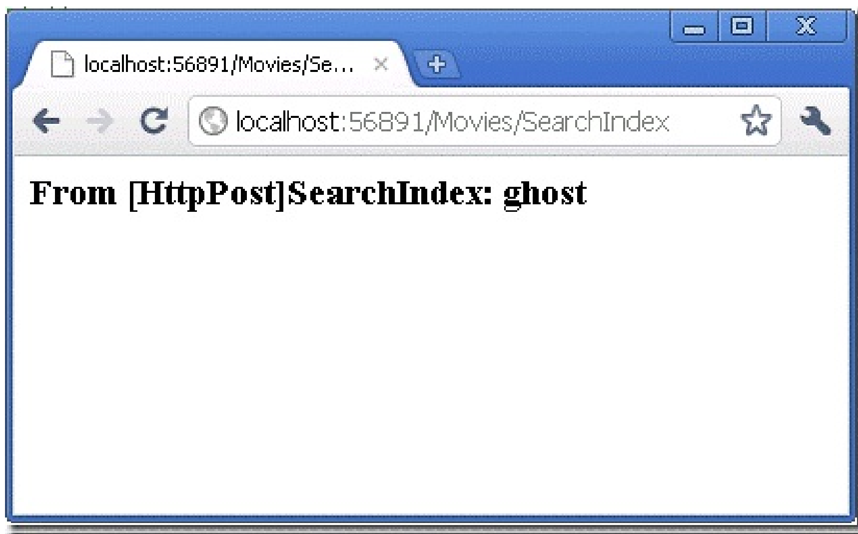
```
[HttpPost]

public string Index(FormCollection fc, string searchString)

{

return "<h3> From [HttpPost]Index: " + searchString + "</h3>";

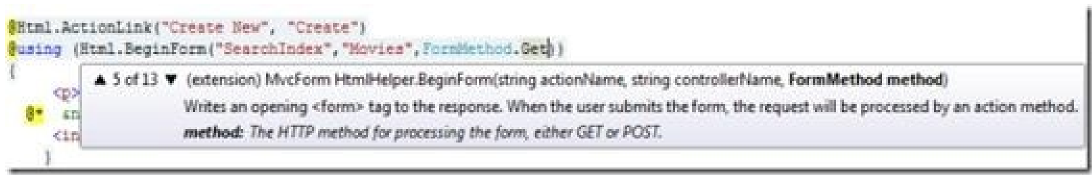
}
```



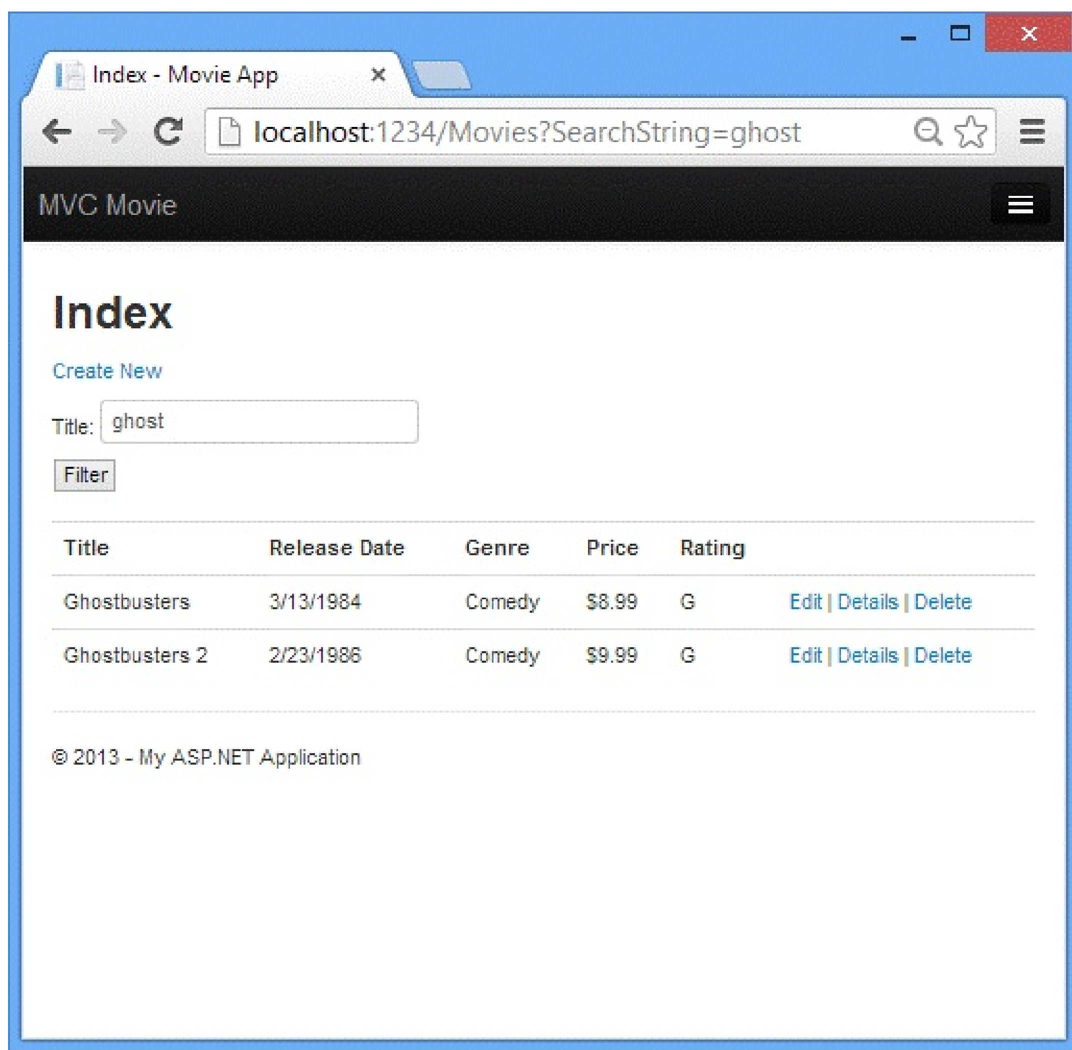
但是，即使您添加此 `HttpPost` Index方法，这一实现其实是有局限的。想象一下您想要添加书签给特定的搜索，或者您想要把搜索链接发送给朋友们，他们可以通过单击看到一样的电影搜索列表。请注意 HTTP POST 请求的 URL 和GET 请求的URL 是相同的（localhost:xxxxx/电影/Index）—— 在 URL 中没有搜索信息。现在，搜索字符串信息作为窗体字段值，发送到服务器。这意味着您不能在 URL 中捕获此搜索信息，以添加书签或发送给朋友。

解决方法是使用重载的BeginForm.aspx)，它指定 POST 请求应添加到 URL 的搜索信息，并应该路由到 HttpGet版的 Index方法。将现有的无参数 `BeginForm` 方法，修改为以下内容

```
@using (Html.BeginForm("Index","Movies",FormMethod.Get))
```



现在当您提交搜索，该 URL 将包含搜索的查询字符串（query string）。搜索还会请求到 `HttpGet` Index操作方法，即使您也有一个 `HttpPost` Index方法。



按照电影流派添加搜索

如果您添加了 `HttpPost` 的Index方法，请立即删除它。

接下来，您将添加功能可以让用户按流派搜索电影。将Index方法替换成下面的代码：

```
public ActionResult Index(string movieGenre, string searchString)
{ var GenreLst = new List<string>(); var GenreQry = from d in db.Movies orderby d.Genre s
GenreLst.AddRange(GenreQry.Distinct());
ViewBag.movieGenre = new SelectList(GenreLst); var movies = from m in db.Movies select m;
{
movies = movies.Where(s => s.Title.Contains(searchString));
} if (!string.IsNullOrEmpty(movieGenre))
{
movies = movies.Where(x => x.Genre == movieGenre);
} return View(movies);
}
```

这个版本的Index方法将接受一个附加的 `movieGenre` 参数。前几行的代码会创建一个 `List` 对象来保存数据库中的电影流派。

下面的代码是从数据库中检索所有流派的 LINQ 查询。

```
var GenreQry = from d in db.Movies orderby d.Genre select d.Genre;
```

该代码使用泛型 List集合的 AddRange方法将所有不同的流派，添加到集合中的。(使用 `Distinct` 修饰符，不会添加重复的流派 -- 例如，在我们的示例中添加了两次喜剧)。

该代码然后在 `ViewBag` 对象中存储了流派的数据列表。的SelectList对象在ViewBag作为存储类数据（这样的电影流派），然后在下拉列表框中的数据访问类别，是一个典型的MVC applications的方法。

下面的代码演示如何检查 `movieGenre` 参数。如果它不是空的，代码进一步指定了所查询的电影流派。

```
if (!string.IsNullOrEmpty(movieGenre))
{
      movies = movies.Where(x => x.Genre == movieGenre);
}
```

如前所述，查询数据不会在数据库上运行，直到电影列表迭代结束（恰发生在View，Index方法返回后）。

Index视图添加标记，以支持按流派搜索电影

在*Views\Movies\Index.cshtml* 文件中，添加Html.DropDownList辅助方法，在TextBox前。完成的代码如下图所示：

```
@model IEnumerable<MvcMovie.Models.Movie> @{
ViewBag.Title = "Index";
} <h2>Index</h2>
<p> @Html.ActionLink("Create New", "Create")
@using (Html.BeginForm("Index", "Movies", FormMethod.Get))
{ <p> Genre: @Html.DropDownList("movieGenre", "All")
Title: @Html.TextBox("SearchString") <input type="submit" value="Filter" />
</p> } </p>
<table class="table">
```

下面的代码：

@Html.DropDownList("movieGenre", "All")

ViewBag 中， "movieGenre" 参考作为key在DropDownList 中搜索 IEnumerable<SelectListItem >. ViewBag填入的操作方法:

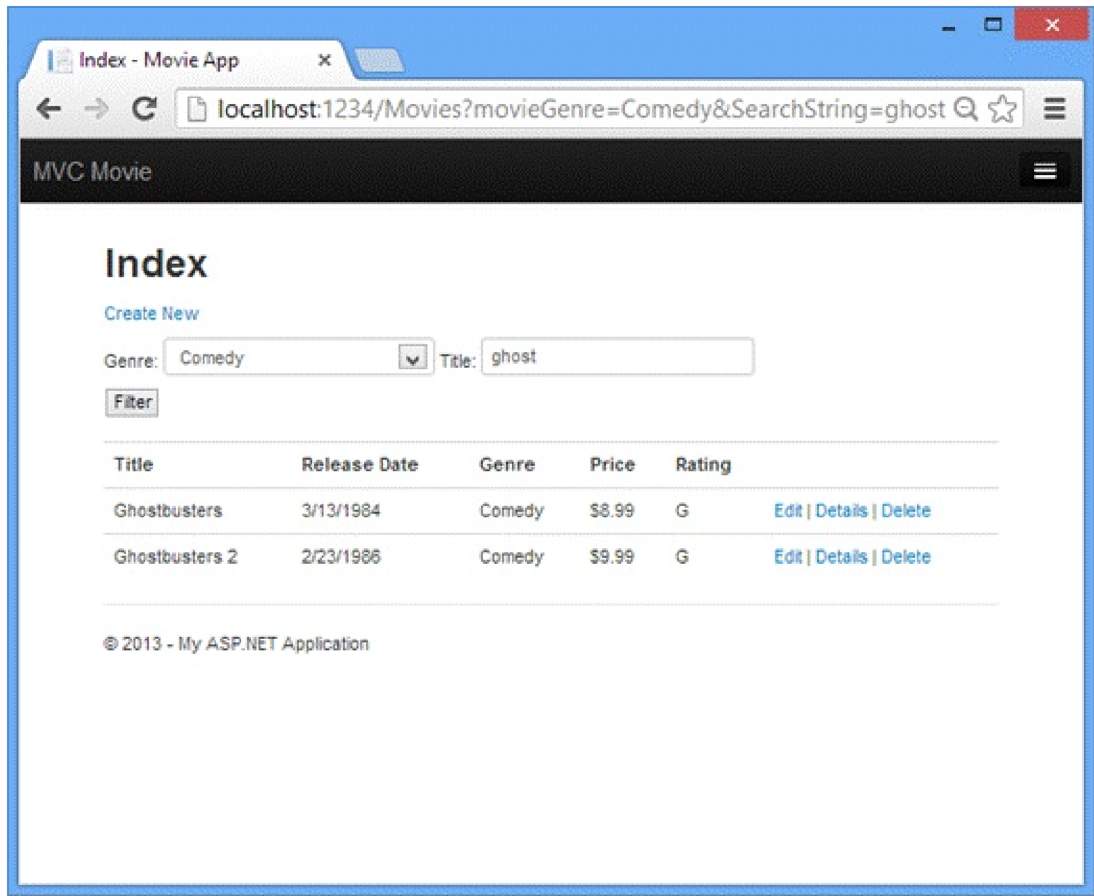
```
public ActionResult Index(string movieGenre, string searchString)
{ var GenreLst = new List<string>(); var GenreQry = from d in db.Movies orderby d.Genre s
GenreLst.AddRange(GenreQry.Distinct());
ViewBag.movieGenre = new SelectList(GenreLst); var movies = from m in db.Movies select m;
{
movies = movies.Where(s => s.Title.Contains(searchString));
} if (!string.IsNullOrEmpty(movieGenre))
{
movies = movies.Where(x => x.Genre == movieGenre);
} return View(movies);
}
```

参数“All”提供的项列表中的预先选择的。如我们使用下面的代码：

```
@Html.DropDownList("movieGenre", "Comedy")
```

在我们的数据库中，我们拥有与“喜剧”流派的电影，“喜剧”在下拉列表中将预先选择。因为我们没有一个电影流派“All”，也没有“All”的SelectList，所以当我们post back后不做任何选择，movieGenre查询字符串值是空的。

运行应用程序并浏览*/Movies/Index*。尝试搜索流派，电影名称，并同时选择这两个条件。



在本节中，您创建了一个搜索的方法和视图，使用它，用户可以通过电影标题和流派来搜索。在下一节中，您将看到如何添加一个属性到Movie model，和如何添加一个初始值设定项值，它会自动创建一个测试数据库。

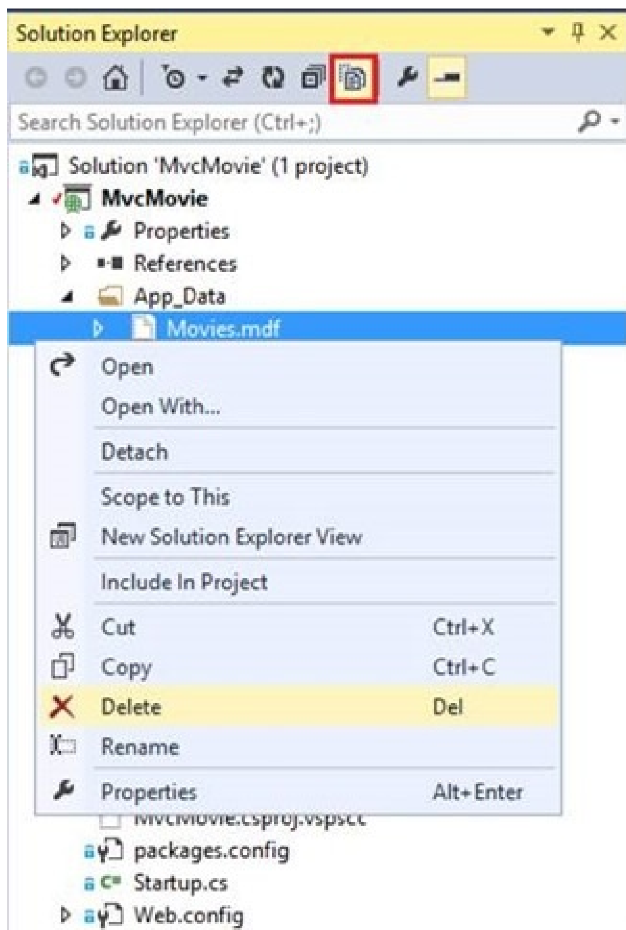
ASP.NET MVC 5 - 给电影表和模型添加新字段

在本节中，您将使用Entity Framework Code First来实现模型类上的操作。从而使得这些操作和变更，可以应用到数据库中。

默认情况下，就像您在之前的教程中所作的那样，使用 Entity Framework Code First自动创建一个数据库，Code First为数据库所添加的表，将帮助您跟踪数据库是否和从它生成的模型类是同步的。如果他们不是同步的，Entity Framework将抛出一个错误。这非常方便的在开发时就可以发现错误，否则您可能会在运行时才发现这个问题。

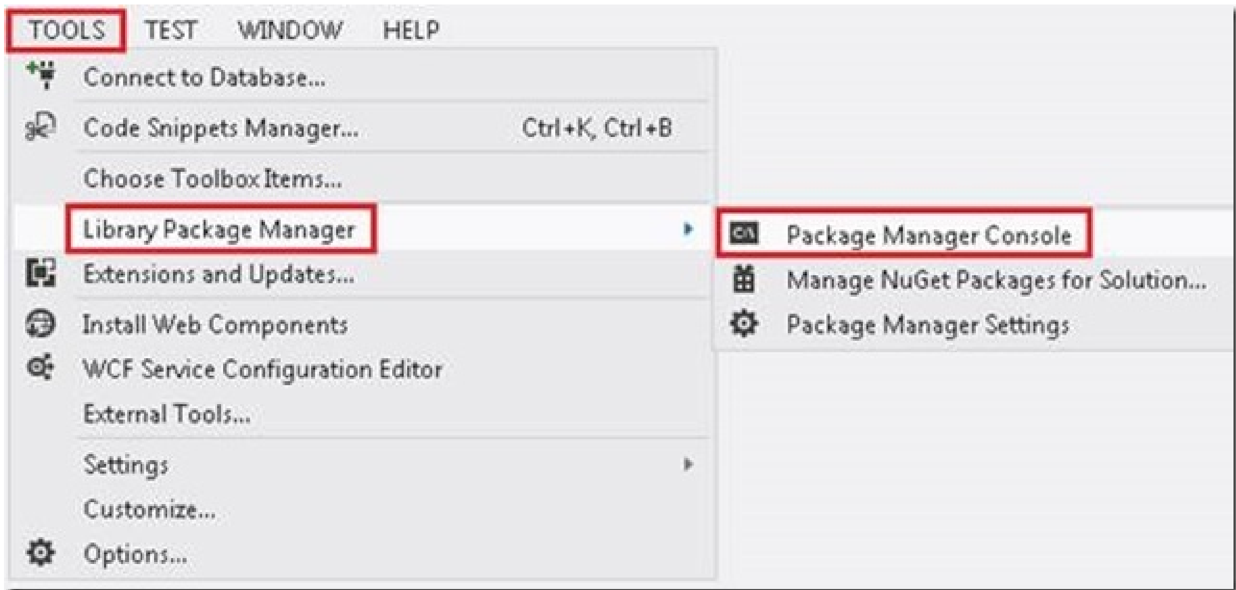
为对象模型的变更设置 Code First Migrations

从解决方案资源管理器中双击*Movies.mdf*，打开数据库工具, 在数据库工具 （数据库资源管理器、 服务器资源管理器或 SQL Server对象资源管理器），右键单击*Movies.mdf* f，并选择删除。



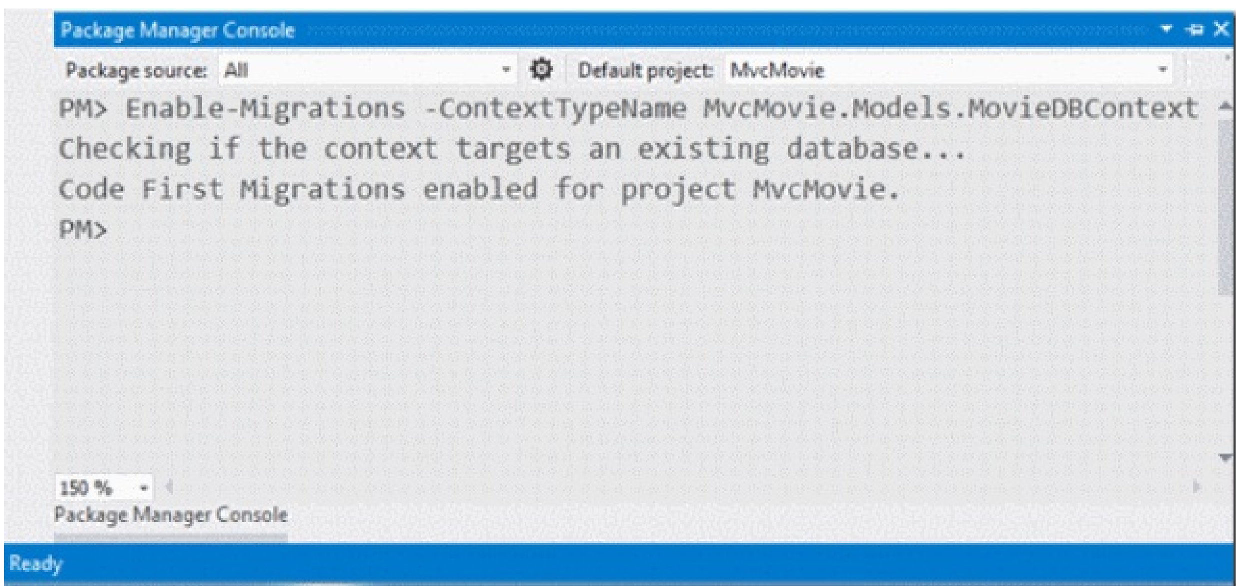
Build应用程序，以确保没有任何编译错误。

从工具菜单上，单击库包管理器，然后点击程序包管理器控制台.

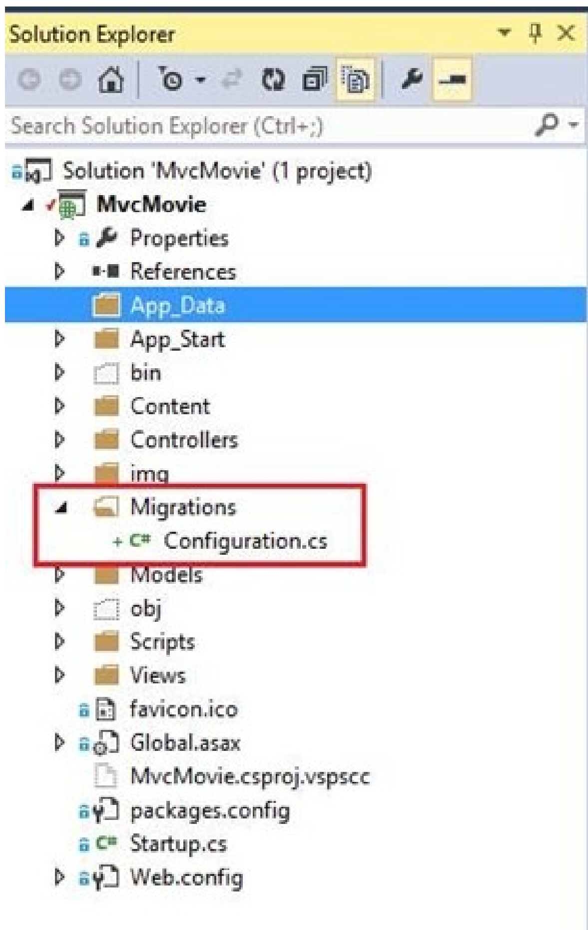


在程序包管理器控制台窗口，在提示符 PM> 后输入：

Enable-Migrations -ContextTypeName MvcMovie.Models.MovieDBContext



如上所示的**Enable-Migrations** 命令，会在*Migrations*文件夹下创建一个*Configuration.cs* 文件。

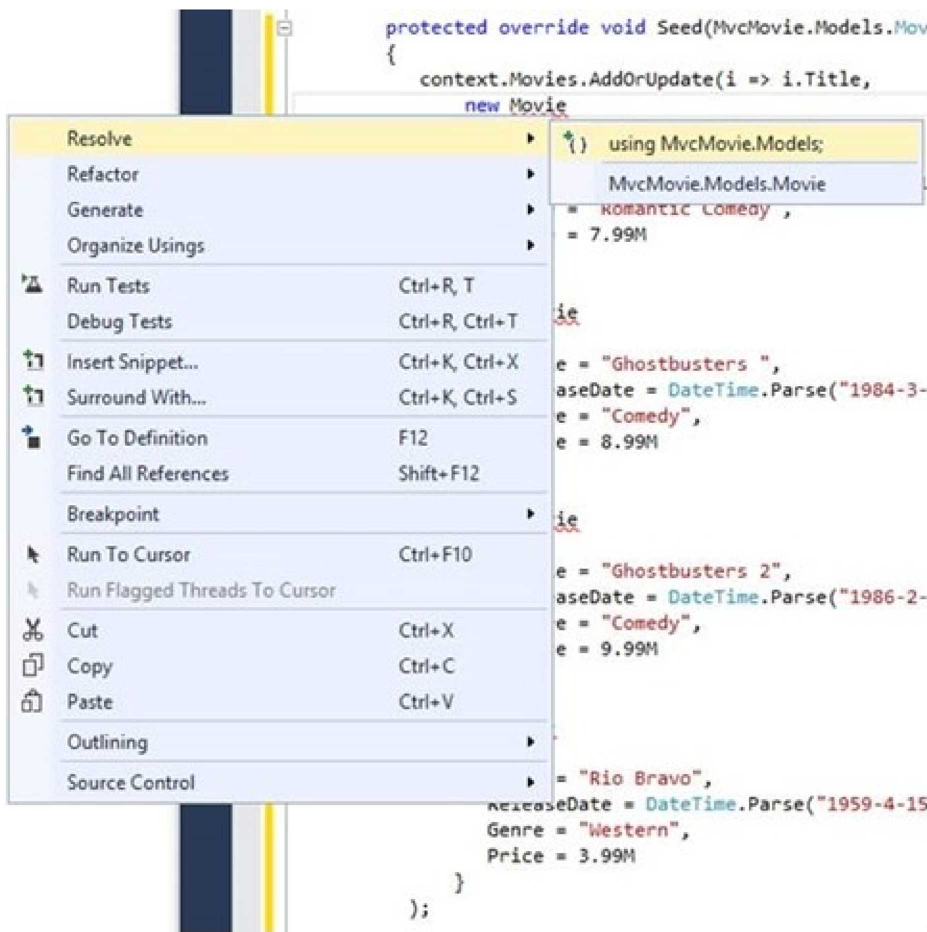


在Visual Studio 里面打开*Configuration.cs* 文件. 用下面的代码替换Seed函数：

```
protected override void Seed(MvcMovie.Models.MovieDBContext context)
{
    context.Movies.AddOrUpdate( i => i.Title, new Movie
        {
            Title = "When Harry Met Sally",
            ReleaseDate = DateTime.Parse("1989-1-11"),
            Genre = "Romantic Comedy",
            Price = 7.99M }, new Movie
         {
             Title = "Ghostbusters ",
             ReleaseDate = DateTime.Parse("1984-3-13"),
             Genre = "Comedy",
             Price = 8.99M }, new Movie
         {
             Title = "Ghostbusters 2",
             ReleaseDate = DateTime.Parse("1986-2-23"),
             Genre = "Comedy",
             Price = 9.99M }, new Movie
       {
           Title = "Rio Bravo",
           ReleaseDate = DateTime.Parse("1959-4-15"),
           Genre = "Western",
           Price = 3.99M }
   );

}
```

右键单击红色波浪线下Movie，并选择"解析(**Resolve**)"，然后单击"**using MvcMovie.Models**;



这样做会增加下面的语句：

using MvcMovie.Models;

Code First Migrations调用Seed的方法，每个迁移（程序包管理器控制台更新数据库），此方法用于updates数据（如果数据存在），或inserted数据。

在AddOrUpdate.aspx)方法在下面的代码执行一个的“upsert”操作：

```
context.Movies.AddOrUpdate(i => i.Title, new Movie
    {
        Title = "When Harry Met Sally",
        ReleaseDate = DateTime.Parse("1989-1-11"),
        Genre = "Romantic Comedy",
        Rating = "PG",
        Price = 7.99M }
```

因为Seed.aspx)方法与每个迁移同时运行时，故，你不能仅仅插入数据，因为当你正试图添加，可能已经完成了创建数据库后的第一次迁移。“upsert”操作阻止错误的发生，如果你尝试插入一个已经存在的行，它覆盖任何数据更改，当你在测试应用程序的同时。你可能不希望这样的事情发生：在某些情况下，当您更改数据测试时，你希望你的变化后数据库同步更新。在这种情况下，你想要做一个有条件的插入操作：只有当它不存在的时候，插入一行。

传递给AddOrUpdate.aspx)的方法的第一个参数， 指定的属性来使用以检查是否已存在某行。对于您所提供的测试影片的数据，Title属性可以被用于此目的，因为每个标题在列表中是唯一：

context.Movies.AddOrUpdate(i => i.Title,

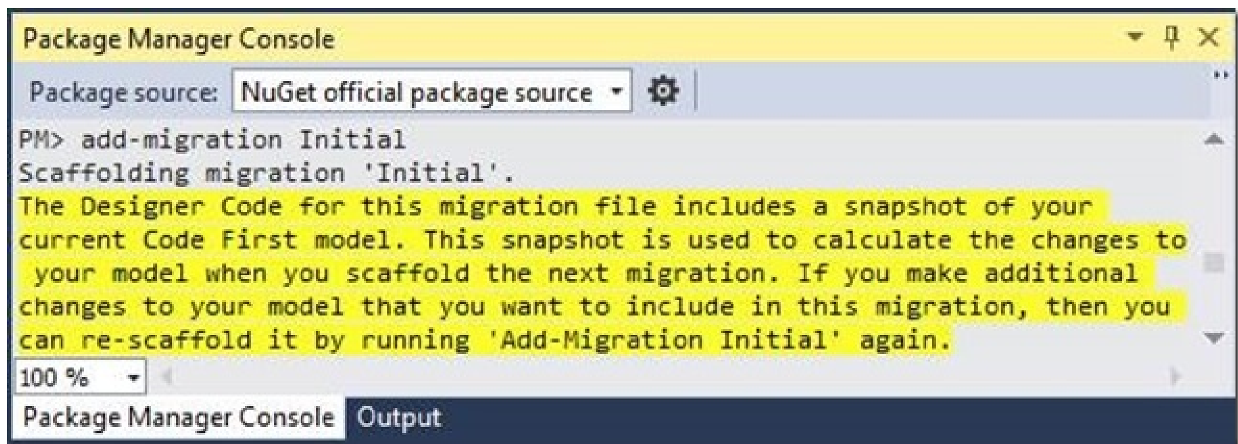
这个代码假设titiles属性是唯一的。如果手动添加一个重复的标题，你会得到下面的异常。

更多关于 AddOrUpdate.aspx) 方法的信息，请参见 Take care with EF 4.3 AddOrUpdate Method..

按 **CTRL-SHIFT-B** 来**Build**工程。（如果此次Build不成功，以下的步骤将会失败。）

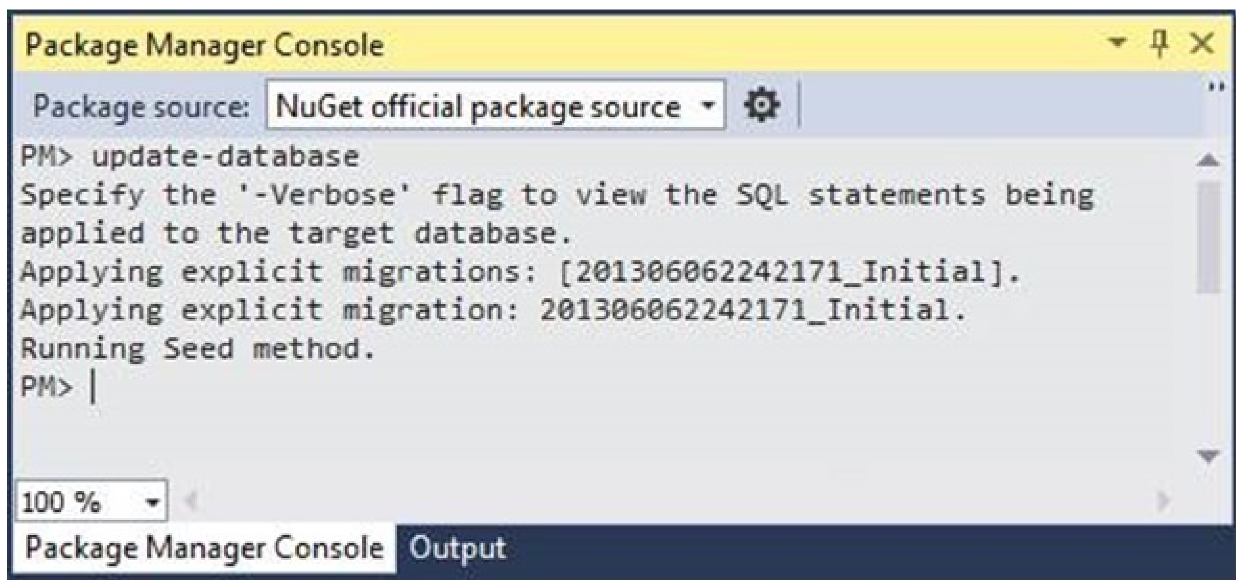
下一步是创建一个DbMigration类，用于初始化数据库迁移。此迁移类将创建新的数据库，这也就是为什么在之前的步骤中你要删除 `movie.mdf` 文件。

在软件包管理器控制台窗口中，输入"add-migration Initial"命令来创建初始迁移。" Initial" 的名称是任意，是用于创建迁移文件的名称。



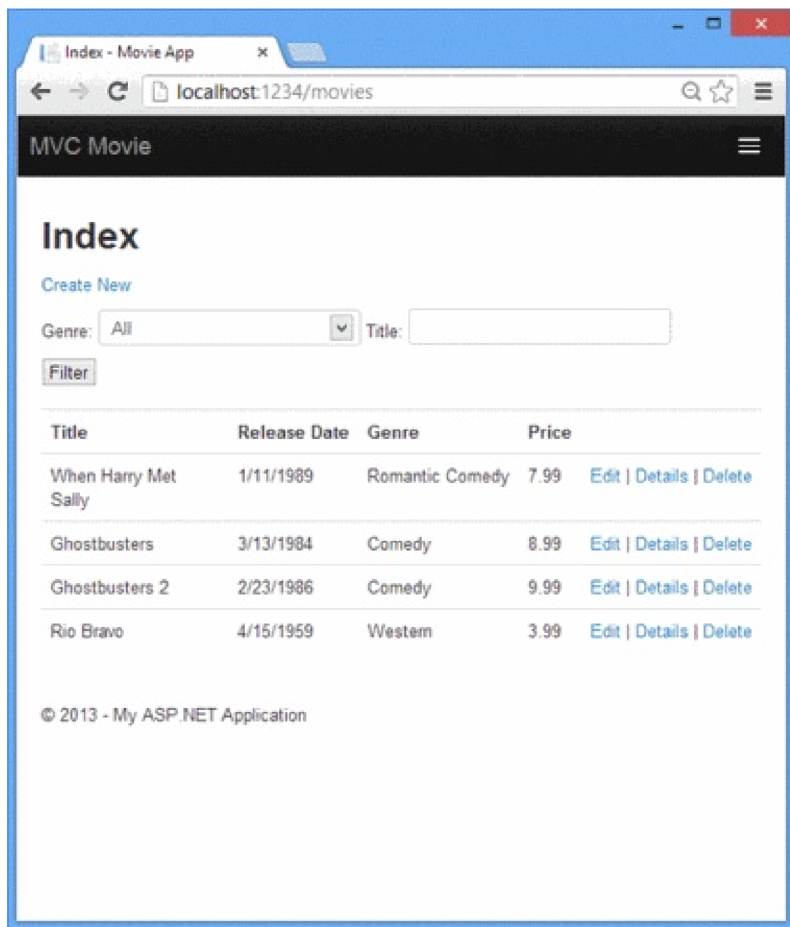
Code First Migrations将会在Migrations文件夹中创建另一个类文件 （文件名为: *{DateStamp}_Initial.cs*)，此类中包含的代码将创建数据库的Schema。迁移文件名使用时间戳作为前缀，以帮助用来排序和查找。查看*{DateStamp}_Initial.cs*文件，它包含了为电影数据库创建电影表的说明。当您更新数据库时， *{DateStamp}_Initial.cs*文件将会被运行并创建 DB 的Schema。然后**Seed**方法将运行，用来填充 DB 的测试数据。

在软件包管理器控制台中，输入命令" update-database "，创建数据库并运行**Seed**方法。



如果您收到表已经存在并且无法创建的错误，可能是因为您已经删除了数据库，并且在执行update-database之前，您运行了应用程序。在这种情况下，再次删除Movies.mdf文件，然后重试update-database命令。如果您仍遇到错误，删除Migration文件夹及其内容，然后从头开始重做。（即删除Movies.mdf文件，然后再进行Enable-Migrations）

运行该应用程序，然后浏览URL /Movies Seed数据显示如下:



为影片模型添加评级(Rating)属性

给现有的Movie类，添加新的Rating属性。打开Models\Movie.cs文件并添加如下Rating属性：

public string Rating { get; set; }

完整的 `Movie` 类如下：

```
public class Movie
{ public int ID { get; set; } public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)] pu
}
```

Build 应用程序 **Build**>**Build Move**或CTRL-SHIFT-B.

因为你已经添加了新的字段，电影类的，你还需要Bind，所以这次新的属性将被包含。更新的绑定属性，Create和Edit动作方法, 包括Rating属性：

[Bind(Include = "ID,Title,ReleaseDate,Genre,Price,Rating")]

您还需要更新视图模板，以显示浏览器视图中创建和编辑新的评级(Rating)属性。

打开*Views\Movies\Index.cshtml*文件，在**Price**列后面添加 `<th>Rating</th>` 的列头。然后添加一个 `<td>` 列来显示 `@item.Rating` 的值。下面是更新的*Index.cshtml*视图模板：

```
@model IEnumerable<MvcMovie.Models.Movie> @{
    ViewBag.Title = "Index";
} <h2>Index</h2>
<p> @Html.ActionLink("Create New", "Create")
    @using (Html.BeginForm("Index", "Movies", FormMethod.Get))
    { <p> Genre: @Html.DropDownList("movieGenre", "All")
        Title: @Html.TextBox("SearchString") <input type="submit" value="Filter" />
    </p> } </p>
<table class="table">
    <tr>
        <th> @Html.DisplayNameFor(model => model.Title) </th>
        <th> @Html.DisplayNameFor(model => model.ReleaseDate) </th>
        <th> @Html.DisplayNameFor(model => model.Genre) </th>
        <th> @Html.DisplayNameFor(model => model.Price) </th>
        <th> @Html.DisplayNameFor(model => model.Rating) </th>

        <th></th>
    </tr> @foreach (var item in Model) { <tr>
        <td> @Html.DisplayFor(modelItem => item.Title) </td>
        <td> @Html.DisplayFor(modelItem => item.ReleaseDate) </td>
        <td> @Html.DisplayFor(modelItem => item.Genre) </td>
        <td> @Html.DisplayFor(modelItem => item.Price) </td>
        <td> @Html.DisplayFor(modelItem => item.Rating) </td>

        <td> @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", new { id=item.ID }) |
            @Html.ActionLink("Delete", "Delete", new { id=item.ID }) </td>
    </tr> } </table>
```

下一步，打开*\Views\Movies\Create.cshtml*文件，并在form标签结束处的附近添加如下代码。您可以在创建新的电影时指定一个电影等级。

```
<div class="form-group"> @Html.LabelFor(model => model.Price, new { @class = "control-lab
                @Html.ValidationMessageFor(model => model.Price) </div>
        </div>

        <div class="form-group"> @Html.LabelFor(model => model.Rating, new { @class = "co
                @Html.ValidationMessageFor(model => model.Rating) </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div> } <div> @Html.ActionLink("Back to List", "Index") </div> @section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
} <div class="form-group"> @Html.LabelFor(model => model.Price, new { @class = "control-l

@Html.ValidationMessageFor(model => model.Price) </div>

</div>

<div class="form-group"> @Html.LabelFor(model => model.Rating, new { @class = "control-la

@Html.ValidationMessageFor(model => model.Rating) </div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Create" class="btn btn-default" />

</div>

</div>

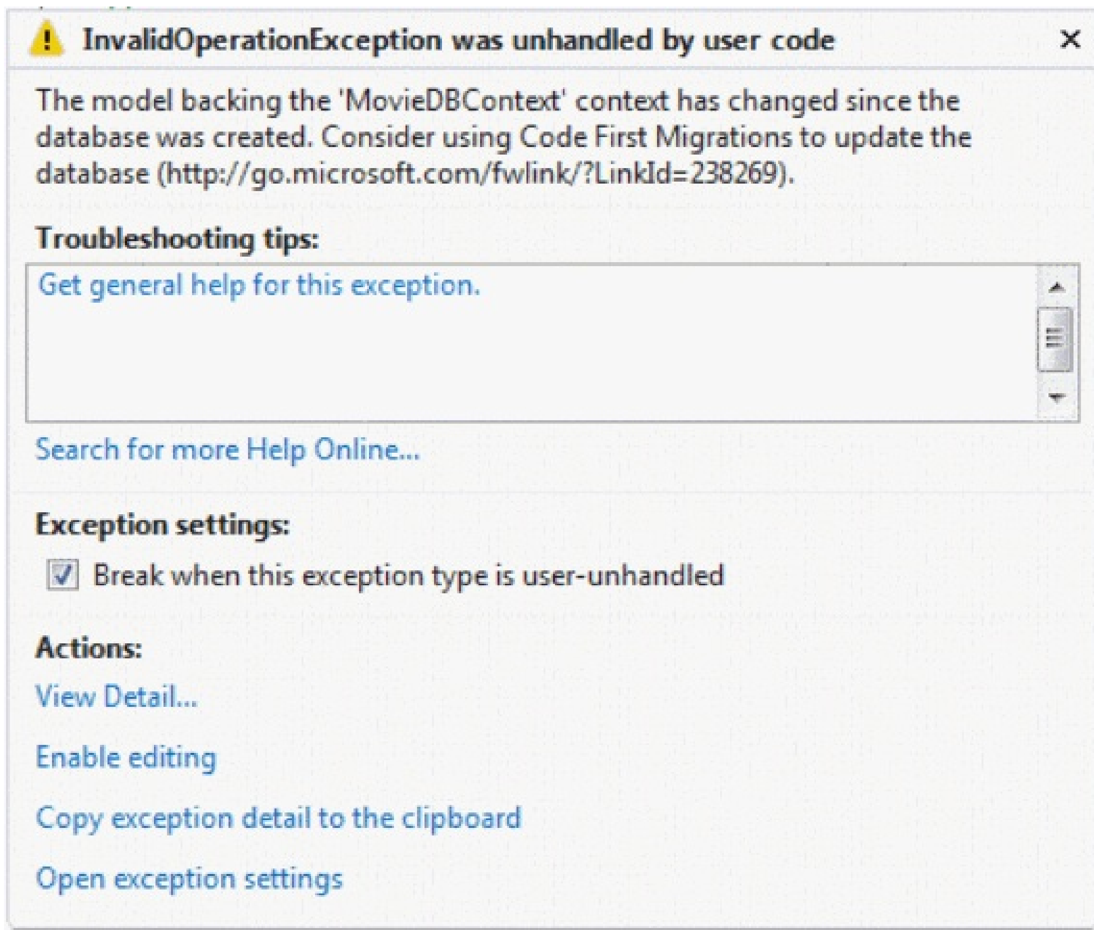
</div> } <div> @Html.ActionLink("Back to List", "Index") </div> @section Scripts {

@Scripts.Render("~/bundles/jqueryval")

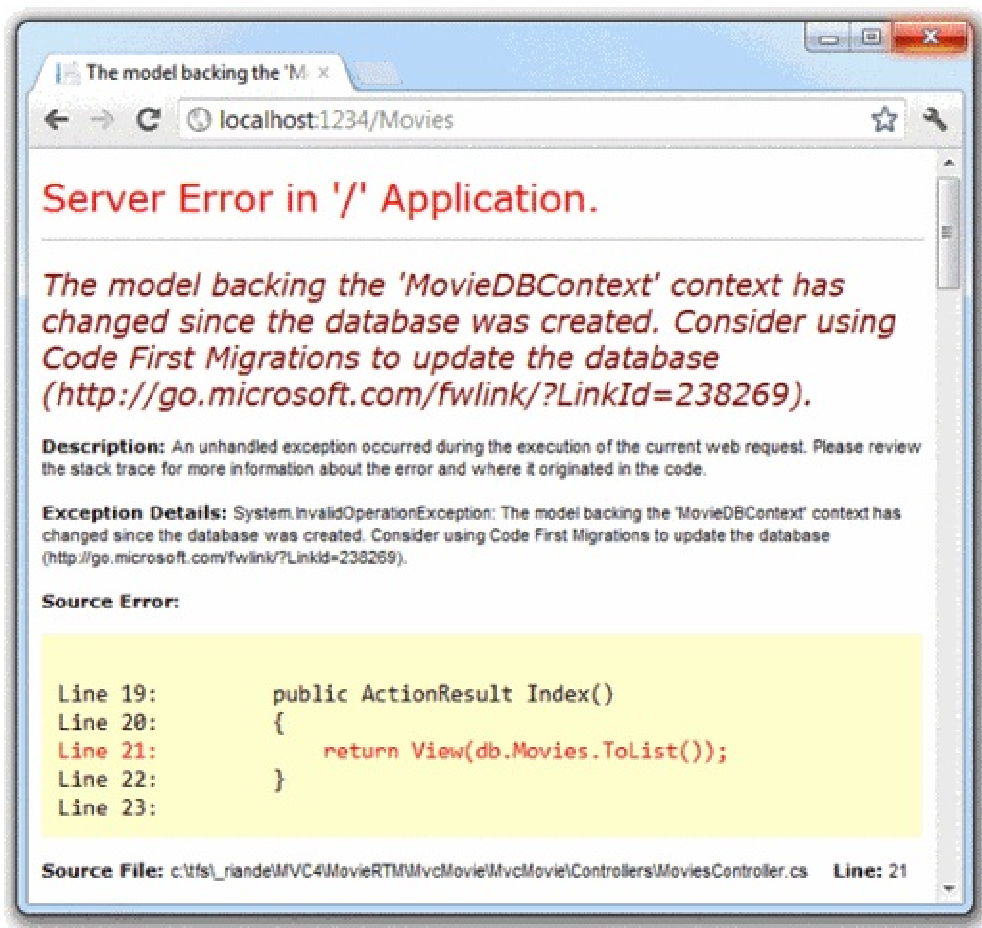
}
```

现在，您已经更新应用程序代码以支持了新的 `Rating` 属性。

现在运行该应用程序，然后浏览 */Movies*的 URL。然而，当您这样做时，您将看到以下之一的错误信息：



自从数据库创建后，备份的'MovieDBContext上下文模型已经改变。请考虑使用Code First Migrations更新数据库 (http://go.microsoft.com/fwlink/?LinkId=238269).



你看到这个错误，因为更新的的Movie模型类中比现在Movie现有数据库表的schema不同。（在数据库表中没有Rating列。）

有几个解决错误的方法:

1. Entity Framework会自动删除并重新创建数据库根据新模型类schema。在开发周期的早期， 这种方式非常方便，当你正在做开发一个测试数据库，它可以让你快速演进模型和数据库schema。不足之处，你将失去现有的数据库中的数据 - 所以对生产数据库你不想使用这种方法！ 通常是一个富有成效的办法，开发一个应用程序来初始化数据库的自动测试数据。更多关于Entity Framework database初始化的信息，请参阅Tom Dykstra's fantastic ASP.NET MVC/Entity Framework tutorial.

2. 显式修改现有数据库的架构，以便它匹配的模型类。这种方法的优点是，你保持你的数据。可以使手动或通过建立数据库更改脚本实现它。

3. 使用Code First Migrations来更新数据库schema。

在本教程中，我们将使用Code First Migrations方法。

更新Seed 方法，以使它可以给新列提供一个值。 打开Migrations\Configuration.cs 文件，在每个Movie下添加Rating 字段.

```
new Movie
        {
            Title = "When Harry Met Sally",
            ReleaseDate = DateTime.Parse("1989-1-11"),
            Genre = "Romantic Comedy",
            Rating = "PG",
            Price = 7.99M },
```

编译解决方案，打开程序包管理器控制台**窗体，输入如下命令：**

add-migration Rating

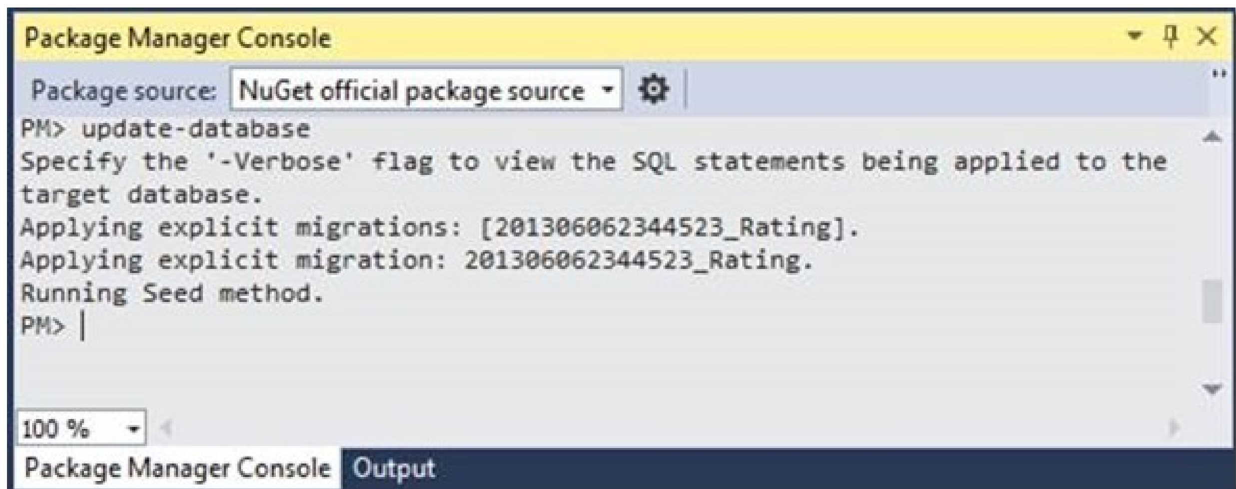
`add-migration` 命令告诉migration framework，来检查当前电影模型与当前的影片 DB Schema并创建必要的代码以将数据库迁移到新的模型。AddRatingMig 是一个任意的文件名参数，用于命名migration文件。它将有助于使得迁移步骤成为一个有意义的名字。

当命令完成后，用Visual Studio 打开类文件，新继承自 `DbMIgration` 类的定义，并在 `Up` 方法中，您可以看到创建新列的代码：

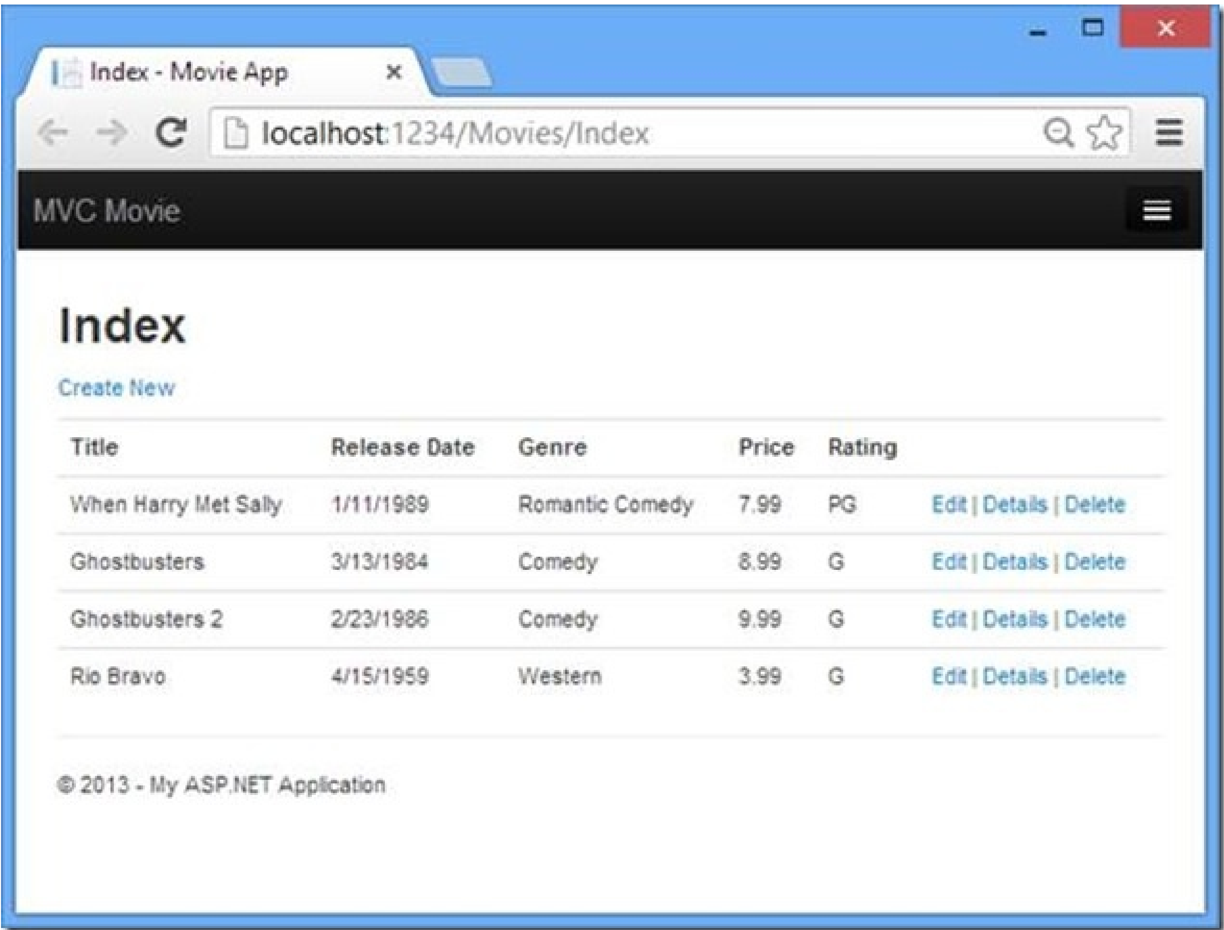
```
public partial class AddRatingMig : DbMigration
{ public override void Up()
    {
        AddColumn("dbo.Movies", "Rating", c => c.String());
    } public override void Down()
    {
        DropColumn("dbo.Movies", "Rating");
    }
}
```

Build解决方案，然后在 程序包管理器控制台窗口中输入"update-database"命令。

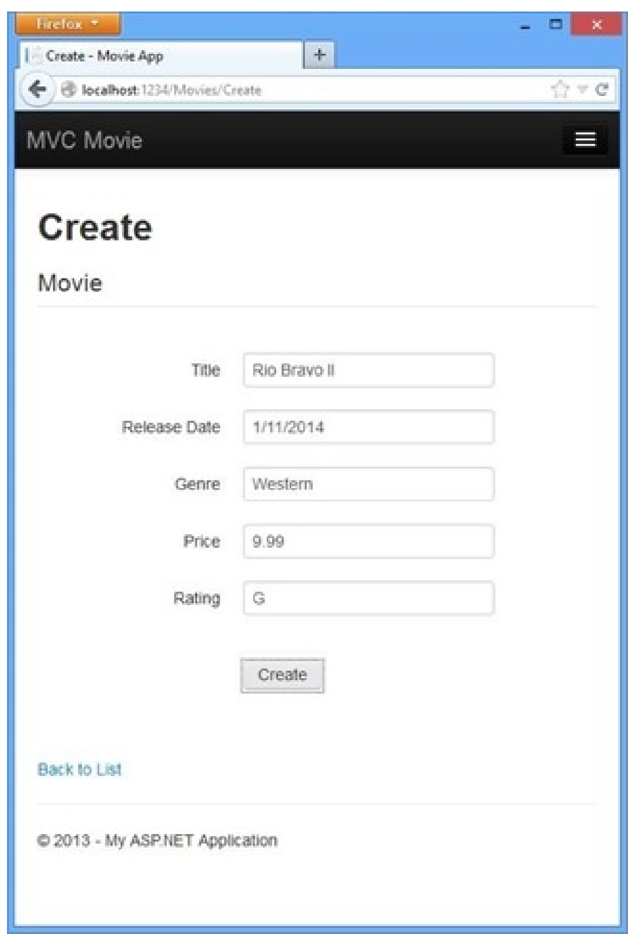
下面的图片显示了 程序包管理器控制台窗口的输出 （日期戳前面添加的评级会有所不同）



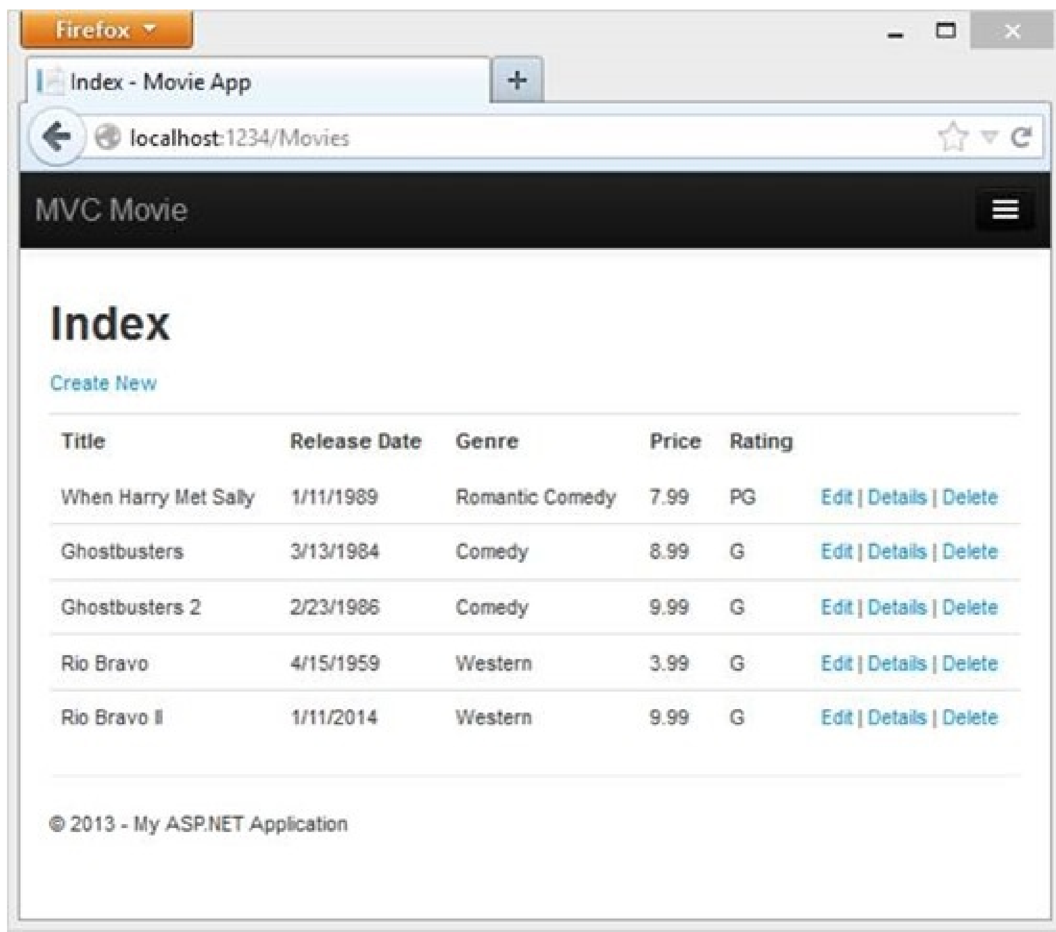
重新运行应用程序，然后浏览 /Movies 的 URL。您可以看到新的评级字段。



单击**CreateNew**链接来添加一部新电影。注意，请您可以为电影添加评级。



单击**Create**。新的电影，包括评级，将显示在电影列表中：



该项目目前正在使用的迁移 (migrations)，当你添加新的字段或更新数据库Schema, 你不需要删除数据库。在下一节中，我们将让更多的架构更改，并使用迁移来更新的数据库。

此外您也应该把 `Rating` 字段添加到Edit、Details和Delete的视图模板中。

您可以再次在 程序包管理器控制台窗口中输入"update-database"命令，将不会有任何新的变化，因为数据库Schema 和模型类现在是匹配的。然而，运行“update-database”将运行再次Seed方法，如果你改变任何种子数据，更改都将丢失，因为Seed的方法upserts数据。你可以阅读更多关于Seed的方法Tom Dykstra's的流行的ASP.NET MVC/Entity Framework tutorial.。

在本节中，您看到了如何修改模型对象并始终保持其和数据库Schema的同步。您还学习了使用填充示例数据来创建新数据库的例子，您可以反复尝试。这只是一个简单的介绍Code First，更完整的教程的请参阅Creating an Entity Framework Data Model for an ASP.NET MVC Application。接下来，让我们看看如何将丰富的验证逻辑添加到模型类，并对模型类执行一些强制的业务规则验证。

ASP.NET MVC 5 - 给数据模型添加校验器

在本节中将会给 `Movie` 模型添加验证逻辑。并且确保这些验证规则在用户创建或编辑电影时被执行。

拒绝重复 DRY

ASP.NET MVC 的核心设计信条之一是DRY: "不要重复自己（DRY --Don’t Repeat Yourself）"。ASP.NET MVC鼓励您指定功能或者行为，只做一次，然后将它应用到应用程序的各个地方。这可以减少您需要编写的代码量，并减少代码出错率，易于代码维护。

给ASP.NET MVC 和 Entity Framework Code First 提供验证支持是 DRY 信条的一次伟大实践。您可以在一个地方 （模型类） 中以声明的方式指定验证规则，这个规则会在应用程序中的任何地方执行。

让我们看看您如何在本电影应用程序中，使用此验证支持。

给电影模型添加验证规则

您将首先向 `Movie` 类添加一些验证逻辑。

打开*Movie.cs* 文件，注意到System.Web 命名空间并未包含System.ComponentModel.DataAnnotations. DataAnnotations提供了一组内置的严重属性，可供您应用于类、属性。(DataAnnotations也包含一个DataType属性，来帮助格式化的办法来校验)

更新 `Movie` 类，以利用内置的 `Required` 、 `StringLength` , RegularExpression和 `Range` 验证属性。以下面的代码为例，以应用验证属性。

```
public class Movie

{ public int ID { get; set; }

[StringLength(60, MinimumLength = 3)] public string Title { get; set; }

[Display(Name = "Release Date")]

[DataType(DataType.Date)]

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)] public

[RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]

[Required]

[StringLength(30)] public string Genre { get; set; }

[Range(1, 100)]

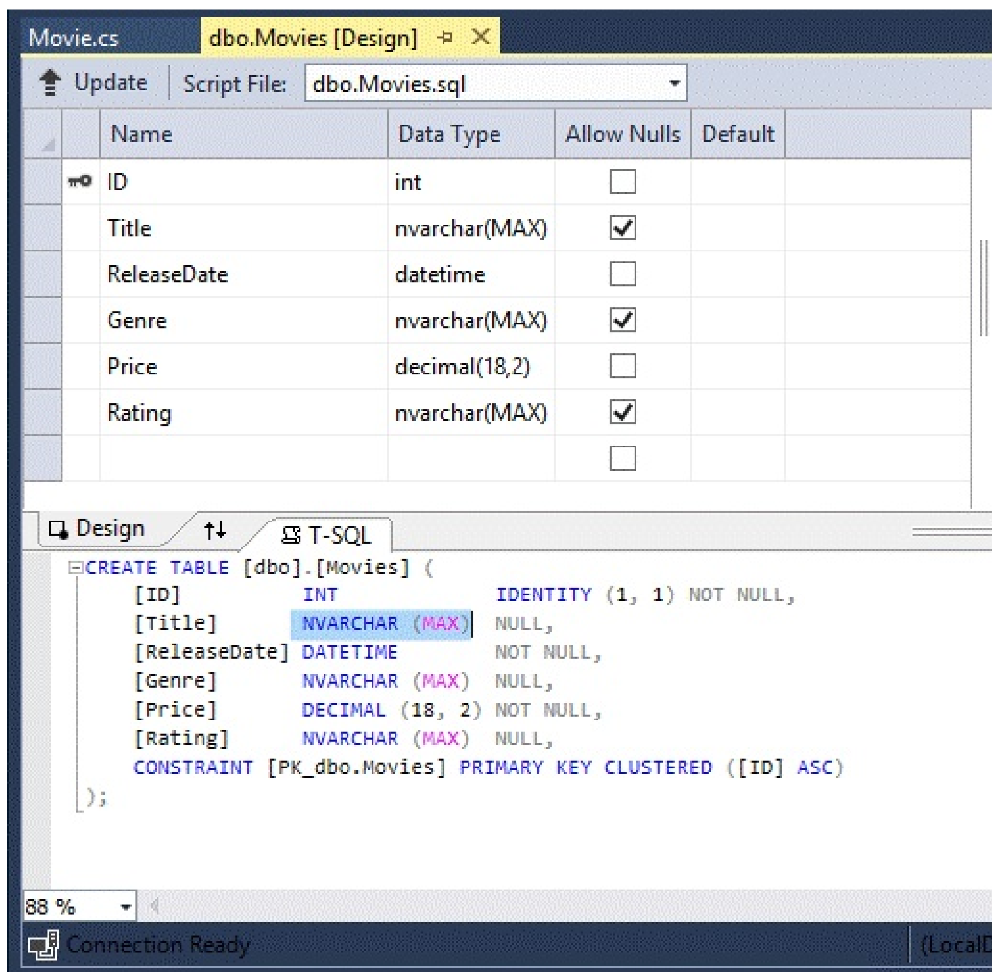
[DataType(DataType.Currency)] public decimal Price { get; set; }

[RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]

[StringLength(5)] public string Rating { get; set; }

}
```

在StringLength属性设置字符串的最大长度，它会在数据库上设置此限制，因此的数据库schema将发生变化。右键单击电影表**, 在服务器资源管理器(Server explorer**)，然后单击打开表定义(**Open Table Definition**)：



在上面的图片中，你可以看到所有的字符串字段被设置为了NVARCHAR (MAX)数据类型. 我们将使用迁移来更新架构。生成解决方案，然后打开软件包管理器控制台(the **Package Manager Console**)，输入如下命令：

add-migration DataAnnotations update-database

当这个命令完成后，Visual Studio将打开类代码文件，它定义了新DbMIgration派生类（DataAnnotations），你可以在Up方法看到更新架构约束代码如下所示：

```
public override void Up()

{

AlterColumn("dbo.Movies", "Title", c => c.String(maxLength: 60));

AlterColumn("dbo.Movies", "Genre", c => c.String(nullable: false, maxLength: 30));

AlterColumn("dbo.Movies", "Rating", c => c.String(maxLength: 5));

}
```

该流派(Genre)字段不再可为Null（也就是说，你必须输入一个值）。该评级（Rating）字段最大长度为5， 标题的最大长度为60。标题(Title)和价格 (Price)的范围的最小长度并没有更改。

请在数据库中，检查电影表的schema：

dbo.Movies [Design]

Update | Script File: dbo.Movies.sql

	Name	Data Type	Allow Nulls	Default
🔑	ID	int	☐	
	Title	nvarchar(60)	☑	
	ReleaseDate	datetime	☐	
	Genre	nvarchar(30)	☐	
	Price	decimal(18,2)	☐	
	Rating	nvarchar(5)	☑	
			☐	

该字符串字段显示新的长度限制和流派字段(Genre)不能再为空。

验证属性指明您想要应用到模型属性的行为。Required.aspx) 和MinimumLength属性指出某一属性不可为空，但没有什么能够阻止用户输入空格来验证。该RegularExpression属性是用来限制哪些字符可以输入。在上面的代码中，流派(Genre)和等级(Rating)只能使用字母（空格，数字和特殊字符是不允许的）。该范围(Range)属性约束的值在一个指定范围内。在StringLength 属性允许您设置一个字符串属性的最大长度，以及最小长度（可选的）。值类型（decimal, int, float, DateTime）有固有必需设置的，不需要的Required属性。

Code First确保你的模型在指定class上在验证规则强制执行之前应用程序将变更储存在数据库中。例如，下面的代码将抛出一个DbEntityValidationException.aspx) 异常时，调用SaveChanges方法时，因为几个必要的Movie属性缺少：

```
MovieDBContext db = new MovieDBContext();

Movie movie = new Movie();

movie.Title = "Gone with the Wind";

db.Movies.Add(movie);

db.SaveChanges(); // <= Will throw server side validation exception
```

上面的代码会抛出以下异常：

Validation failed for one or more entities. 参阅 'EntityValidationErrors' 属性获得更多信息.

具有通过.NET Framework会自动强制执行的验证规则，有助于使你的应用程序更加健壮。它还确保可以不会忘记验证的东西，即在不经意间不会让坏的数据写入数据库。

ASP.NET MVC 的验证错误UI

重新运行应用程序，浏览 */Movies*的 URL。

单击**Create New**链接，来添加一部新电影。在窗体中填写一些无效值，然后单击**Create**按钮。

如同jQuery的客户端验证来检测到错误时，它会显示一个错误消息。

Firefox
Create - Movie App
localhost:1234/Movies/Create
MVC Movie

Create

Movie

Title: Go
The field Title must be a string with a minimum length of 3 and a maximum length of 60.

Release Date: abc
The field Release Date must be a date.

Genre:
The Genre field is required.

Price: 129
The field Price must be between 1 and 100.

Rating: Probably G
The field Rating must be a string with a maximum length of 5.

Create

Back to List

© 2013 - My ASP.NET Application

注意，为了使jQuery支持使用逗号的非英语区域的验证，需要设置逗号（"，"）来表示小数点，如本教程前面所述，你须引入NuGet globalize。请注意，表单在每一个相应的验证错误消息旁边，已经自动使用红色边框的颜色突出显示文本框指明无效数据。这些错误是强制执行了客户端端（使用JavaScript和jQuery）和服务器端（如果用户禁用了JavaScript）。

一个真正的好处是，你并不需要更改MoviesController类或Create.cshtml视图中的一行代码，来启用此验证的用户界面。您在前面教程所创建的控制器和视图会自动启用，使用验证指明的Movie model类的属性。使用Edit行为方法，同样的验证方法也完全适用。直到没有任何客户端验证错误的表单数据，才会被发送回服务器。您可以通过在HTTP POST方法，用一个断点来验证这一点；或通过使用fiddler tool，或者IE浏览器F12 developer tools。

如何验证创建视图和创建方法

您可能很想知道验证用户界面在没有更新控制器或视图代码的情况下是如何生成的。下面列出了 `MovieController` 类中的 `Create` 方法。它们是之前教程中自动生成的，并没有修改。

```
public ActionResult Create()

{ return View();

} // POST: /Movies/Create // To protect from overposting attacks, please enable the speci
 [HttpPost]

[ValidateAntiForgeryToken] public ActionResult Create([Bind(Include = "ID,Title,ReleaseDa

{ if (ModelState.IsValid)

{

        db.Movies.Add(movie);

        db.SaveChanges(); return RedirectToAction("Index");

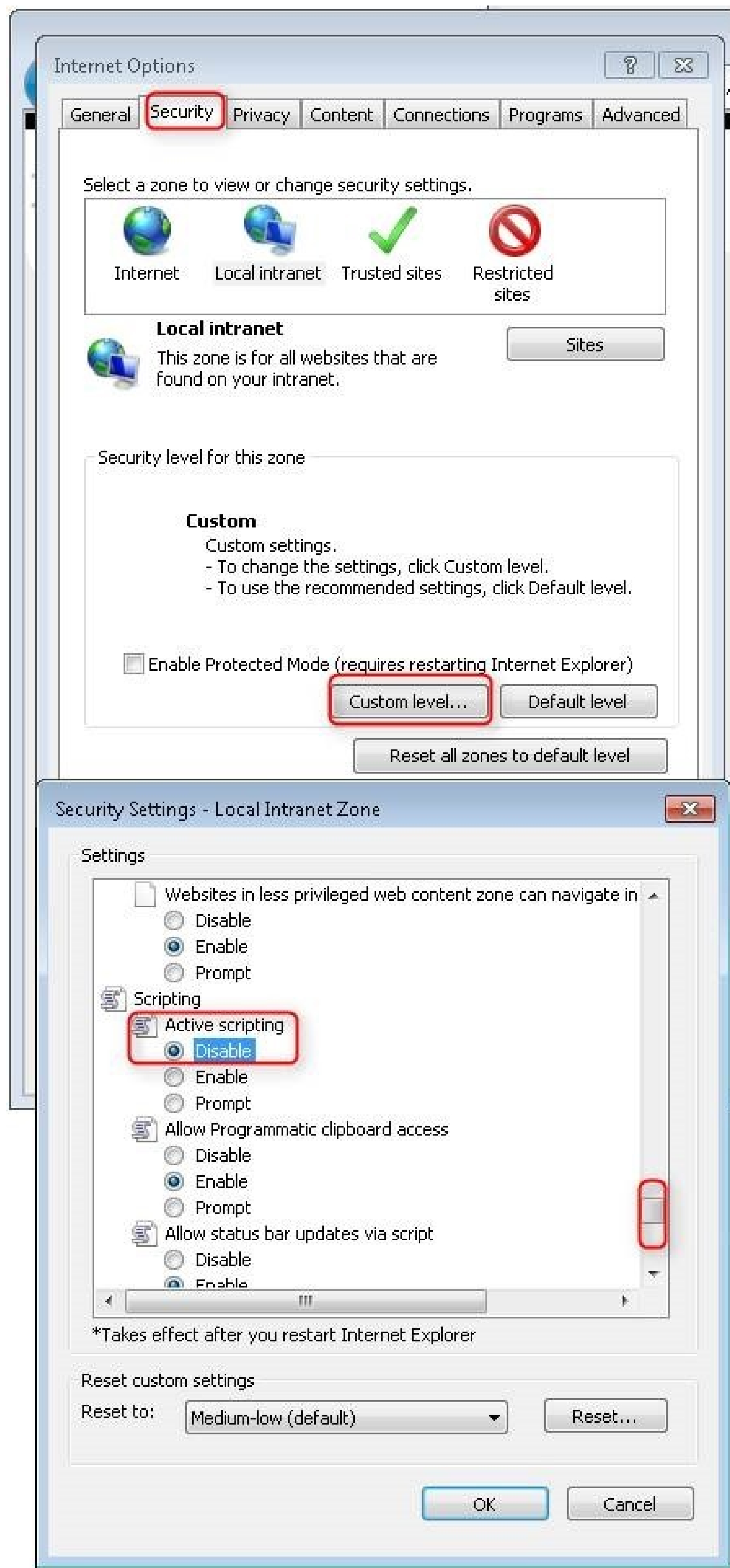
} return View(movie);

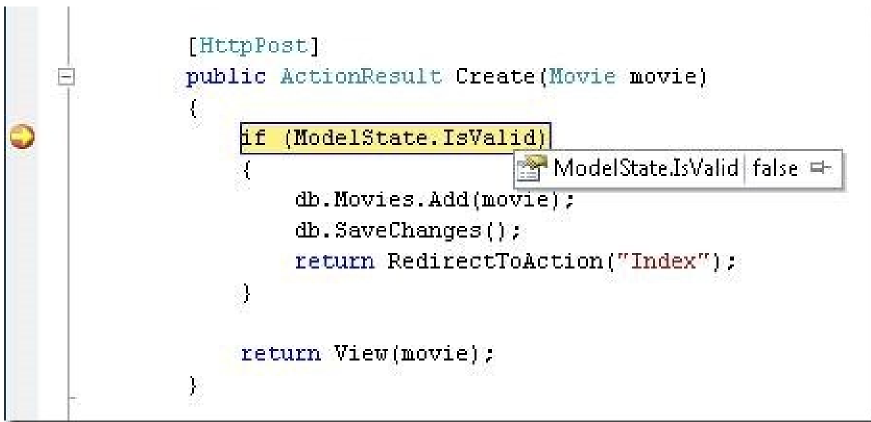
}
```

第一种（HTTP GET） `Create` 方法用来显示初始的创建form。第二个 （ `[HttpPost]` ） 方法处理form的请求。第二种 `Create` 方法 （ `HttpPost` 版本） 调用 `ModelState.IsValid` 来检查是否有任何的Movie验证错误。调用此方法将验证对象上所有应用了验证约束的属性。如果对象含有验证错误，则 `Create` 方法会重新显示初始的form。如果没有任何错误，方法将保存信息到数据库。在我们的电影示例中，我们使用了验证，当客户端检测到错误时，**form不会被post到服务器；所以第二个**Create**方法永远不会被调用**。如果您在浏览器中禁用了JavaScript，客户端验证也会被禁用，HTTP POST `Create` 方法会调用 ModelState.IsValid来检查影片是否含有任何验证错误。

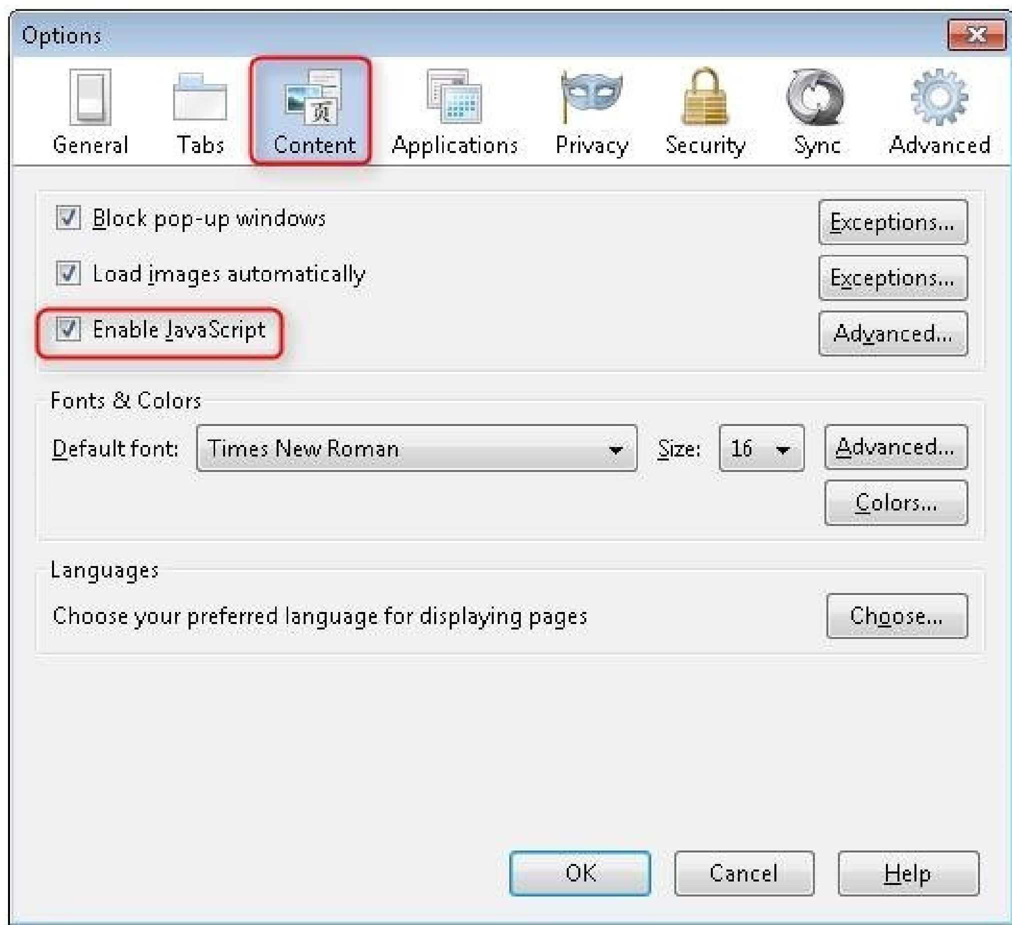
您可以在 `HttpPost Create` 方法中设置一个断点，当客户端验证检测到错误时，不会post form数据，所以永远不会调用该方法。如果您在浏览器中禁用 JavaScript，然后提交具有错误信息的form，断点将会命中。您仍然得到充分的验证，即使在没有 JavaScript的情况下。

下图显示了如何禁用 Internet Explorer 中的 JavaScript。

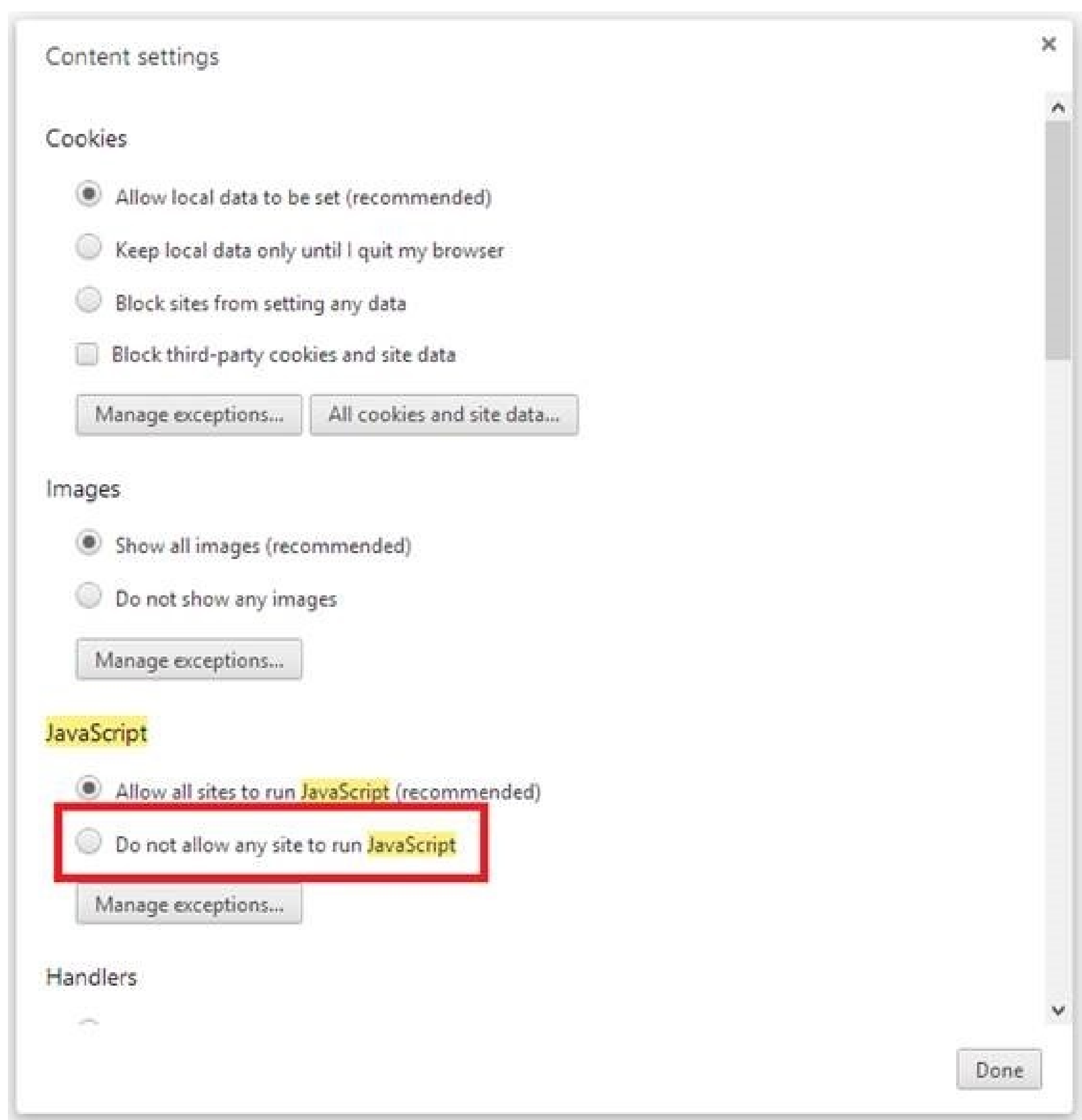

Internet Options
General
Security
Privacy
Content
Connections
Programs
Advanced
Select a zone to view or change security settings.
Internet
Local intranet
Trusted sites
Restricted sites
Local intranet
This zone is for all websites that are found on your intranet.
Sites
Security level for this zone
Custom
Custom settings.
- To change the settings, click Custom level.
- To use the recommended settings, click Default level.
Enable Protected Mode (requires restarting Internet Explorer)
Custom level...
Default level
Reset all zones to default level
Security Settings - Local Intranet Zone
Settings
Websites in less privileged web content zone can navigate in
Disable
Enable
Prompt
Scripting
Active scripting
Disable
Enable
Prompt
Allow Programmatic clipboard access
Disable
Enable
Prompt
Allow status bar updates via script
Disable
*Takes effect after you restart Internet Explorer
Reset custom settings
Reset to:
Medium-low (default)
Reset...
OK
Cancel



下图显示了如何在火狐浏览器中禁用 JavaScript。



下图显示了如何在 Chrome 浏览器中禁用 JavaScript。



下面是框架代码在之前的教程中生成的*Create.cshtml*视图模板。它用来为以上两个操作方法来显示初始的form，同时在验证出错时来重新显示视图。

```
@model MvcMovie.Models.Movie
@{
    ViewBag.Title = "Create";
} <h2>Create</h2> @using (Html.BeginForm())
{
    @Html.AntiForgeryToken() <div class="form-horizontal">
<h4>Movie</h4>
<hr /> @Html.ValidationSummary(true) <div class="form-group"> @Html.LabelFor(model => mod
                @Html.ValidationMessageFor(model => model.Title) </div>
</div> @*Fields removed for brevity.*@ <div class="form-group">
<div class="col-md-offset-2 col-md-10">
<input type="submit" value="Create" class="btn btn-default" />
</div>
</div>
</div> } <div> @Html.ActionLink("Back to List", "Index") </div> @section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

请注意，代码如何使用 `Html.EditorFor` helper 输出为 `Movie` 中的每个属性的 `<input>` 元素。此Helper旁边是对 `Html.ValidationMessageFor` 方法的调用。这两个Helper方法将处理由控制器传递到视图的模型对象（在这里是， `Movie` 对象）。它们会自动查找模型中指定的验证属性，并显示适当的错误消息。

如果您想要在后面更改验证逻辑，您可以做在一个地方，将验证信息添加到模型上。 （此示例中，是 `movie` 类）。您不必担心不符合规则，验证逻辑会在应用程序的不同部分执行——在一个地方定义验证逻辑将会被使用到各个地方。这使代码非常干净，并使它易于维护和扩展。它意味着您会完全遵守DRY原则。

使用DataType属性

打开Movie.cs文件并检查Movie类。在System.ComponentModel.DataAnnotations命名空间提供的格式化（formatting）属性，除了内置的一套验证的属性。我们已经应用了的DataType枚举值的ReleaseDate和Price 字段。下面的代码显示了ReleaseDate和Price 用适当的的DataType属性。

[DataType(DataType.Date)]

public DateTime ReleaseDate { get; set; }

[DataType(DataType.Currency)]

public decimal Price { get; set; }

该DataType属性只提供提示的视图引擎对数据进行格式化（与相应的属性，如<a>取代的URL及 <a href="mailto:EmailAddress.com">取代电子邮件。您可以使用RegularExpression的属性来验证数据格式。DataType属性用于指定一个比数据库内部类型更加具体的一种数据类型，但它们不是验证属性。在这种情况下，我们只需要保留的日期跟踪，而不是日期和时间。该枚举的DataType提供了多种数据类型，如*Date, Time, PhoneNumber, Currency, EmailAddress* 和其他更多的。该的DataType 的属性也可以使应用程序来自动提供特定类型的功能。例如，一个mailto：链接可以DataType.EmailAddress创建和日期选择器可以在支持HTML5的浏览器提供的DataType.Date。该数据类型属性发出的HTML5data-（发音读数据破折号）属性与HTML5的浏览器可以理解。 该DataType 属性不提供任何验证。

DataType.Date并未指定显示的日期格式。默认情况下，根据基于服务器的的CultureInfo.aspx)预设格式显示数据字段。

该DisplayFormat的属性是用来显式地指定日期格式的：

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

public DateTime EnrollmentDate { get; set; }

该ApplyFormatInEditMode设置指定了当值进行编辑显示在一个文本框中，格式化亦应适用。（您可能不希望这样的某些字段 - 例如货币值，你可能不希望在编辑文本框中出现货币符号。）

你可以单独使用DisplayFormat属性；但和DataType属性一起，通常是一个好主意。该DataType 属性传递数据的语义，而不是如何呈现它在屏幕上，并具有以下的优点，不带DisplayFormat的：

· 浏览器可以使HTML5的功能（例如显示一个日历控件，在区域设置相应的货币符号，电子邮件中的链接，等等）。

· 默认情况下，浏览器就会使用基于语言环境(locale)的正确格式呈现数据。

· 在的DataType属性可以使MVC选择合适的字段模板以呈现数据（如果本身所使用的的DisplayFormat使用字符串模板）。欲了解更多信息，请参阅see Brad Wilson's的ASP.NET MVC 2 Templates。 （虽然写的MVC2，本文仍然适用于ASP.NET MVC 5的当前版本。）

如果你使用了的DataType的属性具有一个日期字段，你也必须指明，以确保字段正确地呈现Chrome浏览器中的DisplayFormat属性。欲了解更多信息，请参阅this StackOverflow thread。

注：jQuery的验证不与Range属性和DateTime的同时工作。例如，下面的代码总是显示一个客户端验证错误，即使当日期是在指定的范围内：

[Range(typeof(DateTime), "1/1/1966", "1/1/2020")]

你可能会禁用jQuery的日期校验，而使用带有的Range属性DateTime。这通常不是一个好的做法，在你的模型里，编译器很难确定日期，所以使用Range属性和DateTime效果不好。

下面的代码显示在同一行合并属性：

```
public class Movie
{
public int ID { get; set; }
[Required,StringLength(60, MinimumLength = 3)]
public string Title { get; set; }
[Display(Name = "Release Date"),DataType(DataType.Date)]
public DateTime ReleaseDate { get; set; }
[Required]
public string Genre { get; set; }
[Range(1, 100),DataType(DataType.Currency)]
public decimal Price { get; set; }
[Required,StringLength(5)]
public string Rating { get; set; }
}
```

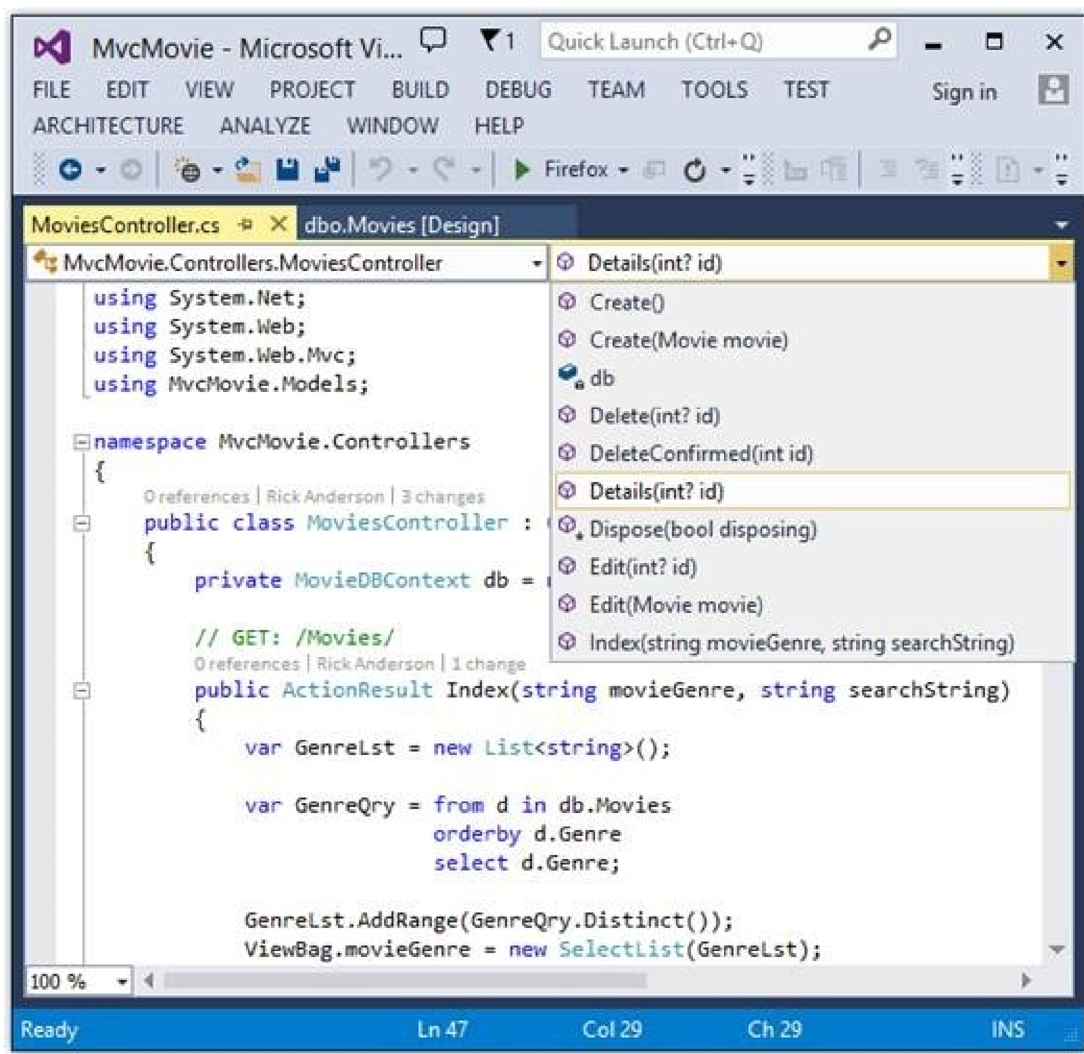
在教程的下一部分，我们先会看看代码，然后再改进一下自动生成的 `Details` 和 `Delete` 方法。

ASP.NET MVC 5 - 查询Details和Delete方法

在这部分教程中，接下来我们将讨论自动生成的Details和Delete方法。

查询Details和Delete方法

打开Movie控制器并查看Details方法。




```
public ActionResult Details(int? id)
{ if (id == null)
{ return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
}
Movie movie = db.Movies.Find(id); if (movie == null)
{ return HttpNotFound();
} return View(movie);
}
```

MVC scaffolding引擎增加了一个注释表明，在调用的HTTP请求方法中，GET请求有三个URL段，Movies控制器，Details方法和ID值。

Code First 使得您可以轻松的使用Find.aspx)方法来搜索数据。一个重要的安全功能内置到了方法中。方法首先验证 `Find` 方法已经找到了一部电影，然后再执行其它代码。例如，黑客可以通过更改*http://localhost:xxxx/Movies/Details/1*到*http://localhost:xxxx/Movies/Details/12345*（或某些其它值，不代表实际影片的值）从而使得链接URL 出现错误。如果您没有检测是否找到了Movie， null Movie会导致出现数据错误。

查看 `Delete` 和 `DeleteConfirmed` 方法。

```
// GET: /Movies/Delete/5

public ActionResult Delete(int? id)

{ if (id == null)

{ return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Movie movie = db.Movies.Find(id); if (movie == null)

{ return HttpNotFound();

} return View(movie);

} // POST: /Movies/Delete/5
 [HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken] public ActionResult DeleteConfirmed(int id)

{

Movie movie = db.Movies.Find(id);

db.Movies.Remove(movie);

db.SaveChanges(); return RedirectToAction("Index");

}
```

请注意， `Delete` 的 `HTTP Get` 方法不会删除指定的电影，它返回删除电影的视图，您可以在此视图中提交 （ `HttpPost` ） 删除电影。如果使用GET 请求执行删除操作（或者执行编辑操作，创建操作或者更改数据的任何其它操作） 开辟了一个安全漏洞。对此的详细信息，请参阅斯蒂芬 · 瓦尔特的博客ASP.NET MVC Tip #46 — Don't use Delete Links because they create Security Holes.

将删除数据的 `HttpPost` 方法命名为唯一签名或名称的 `DeleteConfirmed` 方法。这两个方法的签名如下所示：

```
// GET: /Movies/Delete/5

public ActionResult Delete(int? id) //

// POST: /Movies/Delete/5
 [HttpPost, ActionName("Delete")] public ActionResult DeleteConfirmed(int id)
```

公共语言运行时 (CLR)重载方法时，需要方法具有独特唯一的签名 （方法名称相同但不同的参数列表）。但是，在这里您需要两种删除方法 —— 一个 GET方法和一个POST方法它们都具有相同的签名。（他们都需要接受一个整数作为参数）。

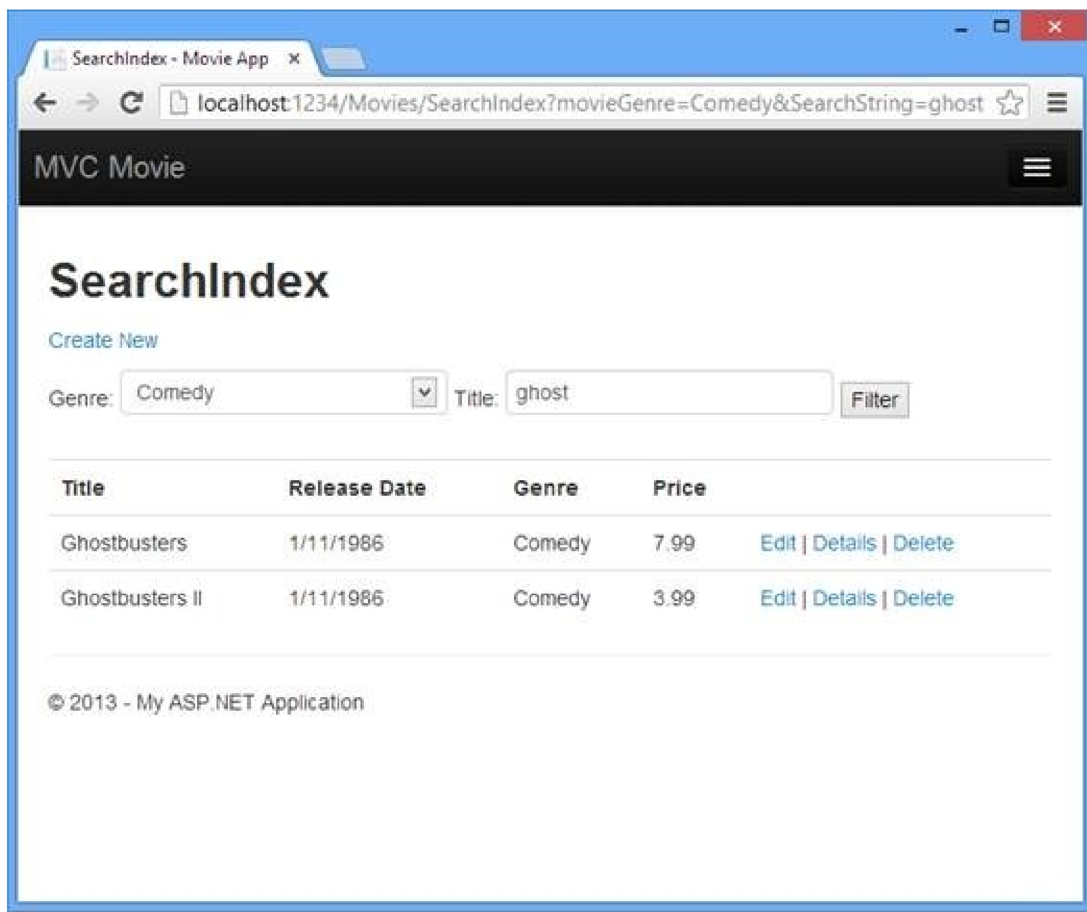
要解决这一点，可以有几种办法。一是使用不同的方法名称。这是框架代码在前面的示例中所使用的方法。然而，这就带来了一个小问题： ASP.NET 将部分的 URL按名称映射到操作方法，如果您重命名了方法，通常Routing将无法找到该方法。解决方法是您在示例中看到的，将 `ActionName("Delete")` 属性添加到 `DeleteConfirmed` 方法。这会有效的执行Routing系统的Url映射，这样一个包含*/Delete/*的 POST 请求的URL 将找到 `DeleteConfirmed` 方法。

另一个常见的方法，来避免具有相同名称和签名的方法，是人为地改变POST 方法，包括未使用参数的签名。例如，有些开发人员添加参数类型 FormCollection，FormCollection是会传递给 POST 方法的，然后根本不使用此参数：

```
public ActionResult Delete(FormCollection fcNotUsed, int id = 0)
{
Movie movie = db.Movies.Find(id); if (movie == null)
{ return HttpNotFound();
}
db.Movies.Remove(movie);
db.SaveChanges(); return RedirectToAction("Index");
}
```

小结

您现在有一个完整的 ASP.NET MVC 应用程序并在本地的 DB 数据库中存储数据。您可以创建、 读取、 更新、 删除和搜索电影。



下一步

在您构建和测试一个Web应用程序之后，下一步就是将其提供给其他人，以使得通过互联网访问。要做到这一点，你需要将它部署到一个Web主机。 如通过微软的free Windows Azure trial account，您可以部署多达10个Web站点。我建议你下一步请按照我的教程Deploy a Secure ASP.NET MVC app with Membership, OAuth, and SQL Database to a Windows Azure Web Site，以更深入了解如何部署。另外，还有一个很好的教程是Tom Dykstra's的中级的Creating an Entity Framework Data Model for an ASP.NET MVC Application. Stackoverflow 和 ASP.NET MVC forums。

提出问题的好地方：StackOverflow的ASP.NET MVC的论坛或者GCDN的Web软件开发讨论区。请关注我们的博客，这样你就可以获得最新教程的更新信息流。

任何意见，欢迎反馈。